SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Stjepan Lojen

Zagreb, 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor: Student:

Prof. dr. sc. Biserka Runje, dipl. ing. Stjepan Lojen

Zagreb, 2016.

Izjavljujem da sam ovaj rad izradio samostalno koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem se roditeljima, sestrama, braći i djevojci na velikoj pomoći, povjerenju i strpljenju koje su mi ukazali tijekom studija, kao i prilikom pisanja diplomskog rada. Također, posebno se zahvaljujem mentorici prof. dr. sc. Biserki Runje te asistentici Amaliji Horvatić Novak.

Stjepan Lojen

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **STJEPAN LOJEN** Mat. br.: 0035186276

Naslov rada na hrvatskom jeziku: **PRIMJENA HEURISTIČKIH METODA U MJERITELJSTVU**

Naslov rada na engleskom jeziku: **THE USE OF HEURISTIC METHODS IN METROLOGY**

Opis zadatka:

Pronalazak optimalnog rješenja za pojedini problem optimizacije nekog procesa često je težak zadatak, pogotovo u slučaju zahtjevnih problema kada je za rješavanje istog potrebno pretražiti veliku količinu mogućih rješenja. U tim situacijama se nerijetko, za postizanja dovoljno dobrog rješenja, koriste heurističke metode.
U radu je potrebno:

- Primijeniti više heurističkih metoda za analizu rezultata dobivenih u postupku mjerenja računalnom tomografijom (CT).
- Analizirati rezultate dobivene različitim heurističkim metodama te komentirati točnost dobivenih rezultata.
- Ocijeniti prikladnost heurističkih metoda za analizu rezultata u mjeriteljstvu.

U tekstu diplomskog rada potrebno je navesti korištenu literaturu i eventualnu pomoć pri izradi.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
29. rujna 2016.	1. prosinca 2016.	7., 8. i 9. prosinca 2016.

Zadatak zadao: Predsjednik Povjerenstva:

Prof. dr. sc. Biserka Runje Prof. dr. sc. Franjo Cajner

SADRŽAJ

SADRŽAJ I
POPIS SLIKA IV
POPIS TABLICA V
POPIS ALGORITAMA VI
POPIS MATLAB KODOVA VII
POPIS KRATICA VIII
POPIS OZNAKA IX
SAŽETAK XI
SUMMARY XII
UVOD 1
1. METODE OPTIMIZACIJE 2
2. HEURISTIČKE METODE OPTIMIZACIJE 4
2.1. Heuristika specifičnih problema 5
2.1.1. Podjela heuristike specifičnih problema 5
2.2. Metaheuristika 6
2.2.1. Podjela metaheuristike 7
3. METAHEURISTIKA - LOKALNE METODE 10
3.1. Metoda lokalnog pretraživanja 10
3.1.1. Odabir susjednog rješenja 11
3.1.2. Izlazak iz lokalnog optimuma 11
3.2. Metoda simuliranog kaljenja 12
3.2.1. Prihvaćanje poteza 14
3.2.2. Raspored hlađenja 14
3.2.2.1. Polazna temperatura 15
3.2.2.2. Ravnotežno stanje 15
3.2.2.3. Hlađenje 16
3.2.2.4. Kriterij zaustavljanja 17
3.3. Tabu pretraživanje 17
3.3.1. Memorija TS algoritma 18
3.3.1.1. Tabu lista 18
3.3.1.2. Srednjoročna memorija 19

3.3.1.3. Dugoročna memorija 20
3.3.2. Kriterij aspiracije 20
3.3.3. Kriterij zaustavljanja 21
3.4. GRASP metoda 21
3.4.1. Korak izgradnje 21
3.4.2. Korak lokalnog pretraživanja 22
3.4.3. Parametar α 22
3.5. Monte Carlo metoda 24
4. METAHEURISTIKA - GLOBALNE METODE 25
4.1. Genetski algoritam 25
4.1.1. Generiranje početne populacije 26
4.1.2. Selekcija 26
4.1.2.1. Kolo sreće 26
4.1.2.2. Rang selekcija 27
4.1.2.3. K-turnirska selekcija 28
4.1.3. Križanje 29
4.1.4. Mutacija 30
4.2. Optimizacija kolonijom mrava 30
4.2.1. Algoritam ACO 32
4.3. Optimizacija rojem čestica 33
4.3.1. Ažuriranje brzine, položaja i rješenja 34
4.3.2. Algoritam PSO 35
4.4. Optimizacija rojem pčela 36
4.4.1. BCO algoritam 37
4.4.1.1. Odluka lojalnosti pčele 38
4.4.1.2. Proces regrutacije 39
4.4.1.3. Primjena BCO-a 39
4.4.2. ABC algoritam 40
5. IMPLEMENTACIJA 43
5.1. Računalna tomografija 43
5.1.1. Primjena ICT-a 44
5.1.1.1. Montažna ili vizualna analiza te dimenzionalna mjerenja 44
5.1.1.2. Usporedba dijelova 44

5.1.1.3. Otkrivanje oštećenja, šupljina, pukotina i kvara 44
5.2. Implementacija heurističkih metoda kod upotrebe metode računalne tomografije . 45
5.2.1. Implementacija GA metode 45
5.2.1.1. Optimiranje parametara funkcije cilja koristeći GA alat 50
5.2.2. Implementacija PSO metode 54
5.2.3. Implementacija ABC metode 58
5.2.4. Usporedba rezultata svih metoda 63
ZAKLJUČAK 65
LITERATURA 66
PRILOZI 68

POPIS SLIKA

Slika 1.1 Podjela metoda optimizacije [4] 2
Slika 2.1 Podjela heurističkih metoda [6, 7] 5
Slika 3.1 Ponašanje LS algoritma u danom okruženju [3] 10
Slika 3.2 Distribucija dobivenog lokalnog optimuma kao funkcija parametra α [3] 23
Slika 4.1 Selekcija kolom sreće [18] 27
Slika 4.2 Situacija prije rangiranja [19] 27
Slika 4.3 Situacija poslije rangiranja [19] 28
Slika 4.4 Primjer jednog od koraka generacijske 3-turnirske selekcije [17] 28
Slika 4.5 Primjer jednog od koraka eliminacijske 3-turnirske selekcije [17] 28
Slika 4.6 Eksperiment ACO-a [21] 31
Slika 4.7 Kretanja čestice i ažuriranje brzine [3] 34
Slika 4.8 Dijagram toka PSO algoritma [7, 25] 36
Slika 5.1 Grafičko sučelje metode GA 47
Slika 5.2 Unos funkcije cilja i ograničenja 50
Slika 5.3 Postavke genetskog algoritma 52
Slika 5.4 Rezultati dobiveni GA metodom 52
Slika 5.5 Promjena iznosa funkcije cilja GA metodom 53
Slika 5.6 Rezultati dobiveni PSO metodom 57
Slika 5.7 Promjena iznosa funkcije cilja PSO metodom 57
Slika 5.8 Rezultati dobiveni ABC algoritmom 62
Slika 5.9 Promjena iznosa funkcije cilja ABC algoritmom 62

POPIS TABLICA

Tablica 3.1 Analogija između fizikalnog sustava i problema optimizacije 13
Tablica 3.2 Memorije TS algoritma 18
Tablica 4.1 Primjer križanja jedinki s jednom točkom prekida 29
Tablica 4.2 Primjer križanja jedinki s više točaka prekida..... 29
Tablica 4.3 Primjer jednostavne mutacije nad slučajno odabranom genu 30
Tablica 5.1 Rezultati svih metoda 63

POPIS ALGORITAMA

Algoritam 3.1 Algoritam lokalnog pretraživanja [3] 11
Algoritam 3.2 Algoritam simuliranog kaljenja [3] 13
Algoritam 3.3 Algoritam tabu pretraživanja [3] 18
Algoritam 3.4 Algoritam GRASP-a [3] 21
Algoritam 3.5 Algoritam Monte Carlo [15] 24
Algoritam 4.1 Algoritam genetskog algoritma [17] 26
Algoritam 4.2 Algoritam optimizacije kolonijom mrava [3] 32
Algoritam 4.3 Algoritam optimizacije rojem čestica [3] 35
Algoritam 4.4 BCO algoritam [24] 37
Algoritam 4.5 ABC algoritam [25] 41

POPIS MATLAB KODOVA

Kod 5.1 Osnovni kod GA algoritma 46
Kod 5.2 Funkcija cilja zapisana u datoteci m-formata 50
Kod 5.3 Kod PSO algoritma 54
Kod 5.4 Kod ABC algoritma 58
Kod 5.5 Kod za RWC selekciju 61

POPIS KRATICA

ABC	*Artificial Bee Colony*	Kolonija umjetnih pčela
ACO	*Ant Colony Optimization*	Optimizacija kolonijom mrava
BCO	*Bee Colony Optimization*	Optimizacija kolonijom pčela
CAD	*Computer Aided Design*	Oblikovanje pomoću računala
CPU	*Central Processing Unit*	Središnja procesorska jedinica
CT	*Computed Tomography*	Računalna tomografija
ENIAC	*Electronic Numerical Integrator And Computer*	Elektronički numerički integrator i računalo
GA	*Genetic Algorithm*	Genetski algoritam
GRASP	*Greedy Randomized Adaptive Search Procedure*	Pohlepno nasumični prilagodljivi postupak pretrage
ICT	*Industrial Computed Tomography*	Industrijska računalna tomografija
LS	*Local Search*	Lokalno pretraživanje
MST	*Minimum Spanning Tree*	Najmanje razapinjuće stablo
PSO	*Particle Swarm Optimization*	Optimizacija rojem čestica
RE	*Reverse Engineering*	Povratno inženjerstvo
RCL	*Restricted Candidate List*	Ograničeni popis kandidata
SA	*Simulated Annealing*	Simulirano kaljenje
TS	*Tabu Search*	Tabu pretraživanje

POPIS OZNAKA

Oznaka	Jedinica	Opis
α	/	Glavni parametar GRASP metode
α_0	/	Parametar
β	/	Specifična konstanta
Δ^+	/	Prosjek povećanja funkcije cilja
Δd_v	mm	Odstupanje CT vrijednosti promjera vanjskog cilindra od referentne vrijednosti
ΔE	J	Razlika u vrijednosti funkcije cilja (energije)
ρ_1, ρ_2	/	Slučajne varijable između 0 i 1
σ	/	Standardno odstupanje vrijednosti funkcije cilja
$\phi_{i,j}$	/	Nasumični broj između 0 i 1
B	/	Veličina populacije umjetnih pčela
b	/	Broj umjetne pčele
BestCost(it)	mm	Najbolja vrijednost funkcije cilja po iteracijama
BestSol	mm	Najbolja, do sada, pronađena vrijednost funkcije cilja
C	/	Brojač nenapuštanja
C_1	/	Kognitivni faktor učenja
C_2	/	Faktor socijalnog učenja
C_b	/	Vrijednost funkcije cilja pčele b
C_{min}, C_{max}	/	Vrijednosti djelomičnih/cjelovitih rješenja vezanih uz minimalnu i maksimalnu vrijednost funkcije cilja
D	/	Vektor dimenzija D
E	J	Vrijednost funkcije cilja (energije)
fit_i	/	Vrijednost funkcije cilja položaja i
fval	/	Najbolja vrijednost funkcije cilja
GlobalBest	mm	Najbolja, do sada, pronađena vrijednost funkcije cilja
k	/	Boltzmannova konstanta
L	/	Sljedeći broj prijelaza
lb	/	Donja granica nezavisnih varijabli
MaxIt	/	Maksimalni broj iteracija
MCN	/	Maksimalni broj ciklusa
m	/	Faktor geometrijskog povećanja
m_1, m_2	/	Brojevi rješenja koja se povećavaju ili smanjuju u preliminarnim eksperimentima
N	/	Broj pojedinaca (kromosona) u populaciji
NC	/	Parametar kojim se definira učestalost razmjene informacija između pčela
nPop	/	Veličina populacije pčela/čestica
nVar	/	Broj nezavisnih varijabli

p	%	Vjerojatnost prihvaćanja
$p_{\mathrm{b}}^{\mathrm{u+1}}$	%	Vjerojatnost da je pčela *b* lojalna svom prethodnom rješenju
p_{i}	/	Najbolji položaj ostvaren od čestice *i*
p_{g}	/	Najbolji položaj ostvaren od cijelog roja čestica
Q_{b}	/	Normalizirana vrijednost funkcije cilja pčele *b*
Q_{k}	/	Normalizirana vrijednost funkcije cilja *k*-tog reklamiranog rješenja
Q_{max}	/	Maksimalna normalizirana vrijednost funkcije cilja
R	/	Uniformni slučajni broj između 0 i 1
RE	/	Veličina populacije pčela regrutera
s	/	Rješenje
s_0	/	Polazno rješenje
SN	/	Veličina populacije zaposlenih pčela
T	K, °C	Temperatura
T_0	K, °C	Polazna temperatura
T_{F}	K, °C	Konačna temperatura
T_{i}	K, °C	Temperatura pri *i*-toj iteraciji
T_{max}, T_{min}	K, °C	Maksimalna/minimalna temperatura
U	kV	Napon izvora rendgenskih zraka
u	/	Brojač fazi unaprijed
ub	/	Gornja granica nezavisnih varijabli
v_{i}	m/s	Brzina čestice *i*
w	/	Koeficijent inercije
x	/	Koordinate najbolje postignute vrijednost funkcije cilja
x_{i}	/	Položaj čestice *i*

SAŽETAK

Tema je ovog rada dati pregled i podjelu heurističkih metoda, odabrati tri različite metaheurističke metode, provesti optimizaciju problema (funkcije cilja) te na kraju usporediti i komentirati dobivene rezultate. Rad je podijeljen u pet poglavlja. U prvom je poglavlju dana definicija pojma optimizacije te podjela i usporedba egzaktnih i aproksimativnih metoda. U drugom je poglavlju dan pregled heurističkih metoda, koje su podijeljene na heuristiku specifičnih problema te na metaheuristiku. Nakon definirane razlike između tih dviju vrsta heurističkih metoda, dana je njihova detaljnija podjela. U trećem poglavlju opisane su neke od najpopularnijih lokalnih metaheurističkih metoda optimizacije, kao što su: metoda lokalnog pretraživanja, metoda simuliranog kaljenja, tabu pretraživanje, GRASP metoda te metoda Monte Carlo. U poglavlju nakon toga opisane su neke od najpopularnijih globalnih metaheurističkih metoda optimizacije, kao što je genetski algoritam, optimizacija kolonijom mrava, optimizacija rojem čestica te optimizacija rojem pčela. Sljedeće poglavlje (poglavlje 5) zapravo je srž ovoga rada jer su u njemu implementirane metaheurističke metode kroz softver Matlab - *trial version*. Ovo poglavlje se sastoji od: objašnjenja pojma računalne tomografije i industrijske računalne tomografije, prikaza i opisa jednadžbe funkcije cilja te njenih ograničenja, implementacije metode genetskog algoritma, metode optimizacije rojem čestica i metode optimizacije kolonijom umjetnih pčela te usporedbe dobivenih rezultata.

Ključne riječi: heurističke metode, metaheurističke metode, Matlab - *trial version*, industrijska računalna tomografija, genetski algoritam, optimizacija rojem čestica, optimizacija rojem pčela, usporedba dobivenih rezultata

SUMMARY

The aim of this work is to give a review of the division of heuristic methods, to choose three different metaheuristic methods and to carry out the optimization of the problem (objective function) and, finally, to compare and comment on the results. The paper is divided into five chapters. The first chapter introduces the definition of optimization and classification and the comparison of exact and approximate methods. The second chapter gives an overview of heuristic methods, which are divided into heuristics of specific problems and Metaheuristics. After the difference between these two types of heuristic methods has been defined, the paper presents the division in more detail. The third chapter describes some of the most popular local metaheuristic optimization methods, such as local search method, method of simulated annealing, tabu search, GRASP method and Monte Carlo method. The fourth chapter describes some of the most popular global metaheuristic optimization methods, such as genetic algorithm, ant colony optimization, particle swarm optimization and bee swarm optimization. The next chapter (Chapter 5) is the core of this work because it implements metaheuristic methods through software Matlab - trial version. This section consists of a brief explanation of the concept of computed tomography and industrial computed tomography, an illustration and description of the equation objective function and its limitations, implementation of the genetic algorithm, particle swarm optimization and artificial bee colony optimization and by comparing the results obtained.

Keywords: heuristic methods, metaheuristic methods, Matlab - trial version, industrial computed tomography, genetic algorithm, particle swarm optimization, bee colony optimization, comparing the results obtained

UVOD

Optimizacija, riječ koja potječe od latinske riječi optimum, što znači najbolje, predstavlja skup najpovoljnijih uvjeta ili okolnosti, najviše što se može postići - vrhunsko postignuće. Optimizacija je u inženjerskom smislu postupak maksimiziranja ili minimiziranja nekog cilja ili ciljeva u okvirima raspoloživih resursa, odnosno uz zadovoljenje danih ograničenja koja objektivno postoje. U ovome smislu je svaka odluka koju donosimo, te svaka svjesna radnja koju obavljamo, u stvari rezultat nekog svjesnog ili nesvjesnog misaonog procesa optimiranja. U užem smislu, optimizacija je grana matematike i računalne znanosti. Pri tome su problem, skup mogućih rješenja i način procjene kvalitete svakog potencijalnog rješenja formalno definirani, najčešće jezikom matematike.

Razvojem procesora (eng. CPU - *Central Processing Unit*) [1], sve veći broj problema postao je rješiv. Ipak, i uz veoma brz rast procesorske snage računala, teorijska računalna znanost prepoznala je razrede problema koji se danas poznatim matematičkim algoritmima ne mogu riješiti u razumnom vremenu. Mnogi problemi s kojima se svakodnevno susrećemo ubrajaju se u, takozvane, zahtjevne probleme te je za njihovo uspješno rješavanje potrebno izdvojiti dosta vremena i truda. Za te probleme optimum je redovito nepoznat i najbolje što nam suvremena računalna znanost pruža jesu postupci optimizacije koji pronalaze rješenja koja su blizu ili vrlo blizu optimalnog rješenja. Takve se metode zovu aproksimativne metode, u koje se ubrajaju i heurističke metode. Heurističke metode su danas dovoljno usavršene da rutinski rješavaju iznimno zahtjevne probleme usmjeravanja vozila ili mrežnog prometa, probleme parkiranja, planiranja, izrade radnih sati i drugo.

U ovome radu dana je podjela metoda optimizacije, detaljni pregled heurističkih postupaka optimizacije te upotreba, mogućih primjenjivih metoda, na konkretan problem iz prakse. Konkretno govoreći, radi se o primjeni više različitih metaheurističkih metoda na analizu rezultata dobivenih u postupku mjerenja računalnom tomografijom, usporedbi rezultata optimizacije dobivenih različitim metaheurističkim metodama te danim završnim komentarom na temelju dobivenih rezultata.

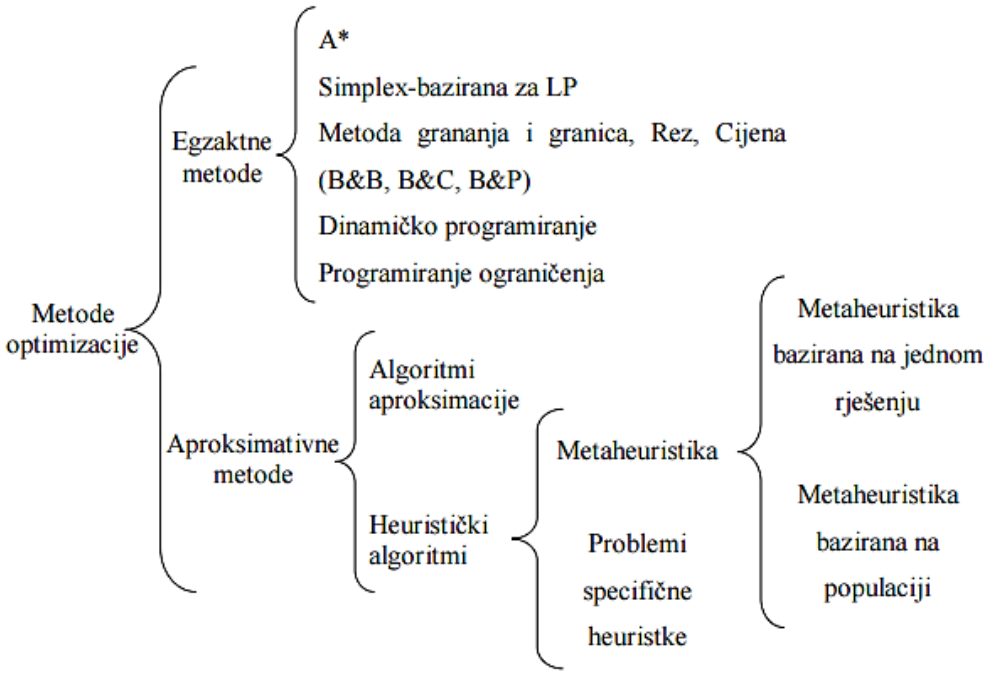
1. METODE OPTIMIZACIJE

Pojam optimizacije znači dobivanje najboljeg rezultata (izlaza) uzimajući u obzir dane okolnosti (ulaze). Optimizacija je najznačajnija aktivnost u procesu donošenja odluke tijekom faze konstruiranja, proizvodnje, održavanja, izgradnje i/ili bilo koje druge inženjerske i menadžerske aktivnosti. Odavde proizlaze dva glavna zahtjeva koja se postavljaju nad metodama optimizacije, a to su [2]:

- kako minimizirati potreban trud kako bi se došlo do željenog cilja,
- te kako maksimizirati željenu korist.

Ako potreban trud ili željena korist za praktičnu situaciju mogu biti kvantizirani i izraženi kao funkcija određene varijable na temelju koje odluka mora biti donesena, onda se smatra da je problem u optimizaciji. Stoga, optimizacija može biti definirana kao proces traženja najboljeg rješenja u formi minimiziranja potrebnog truda (npr. funkcija troškova) ili maksimiziranja željene koristi (npr. funkcija profita), izraženih u obliku funkcija varijabli odluke pod određenim ograničenjima i/ili danim okolnostima. Ako se korist ili potreban trud prikaže kao funkcija $f(x)$, onda se optimizacijski problem prikazuje kao:

- maksimizirati $f(x)$, za $x = x^*$ - gdje $f^*(x)$ predstavlja najveću vrijednost funkcije $f(x)$,
- minimizirati $f(x)$, za $x = x^*$ - gdje $f^*(x)$ predstavlja najmanju vrijednost funkcije $f(x)$.



Slika 1.1 Podjela metoda optimizacije [4]

Prema autoru El-Ghazali Talbi-u [3] glavna podjela metoda optimizacije je na egzaktne metode i aproksimativne metode (Slika 1.1).

Egzaktne metode garantiraju optimalno rješenje približnog, ali strogo definiranog matematičkog modela realnog problema, ne vodeći računa pritom o utrošku vremena. Podjela egzaktnih metoda je na: A* obitelj algoritama traženja, dinamičko programiranje, simpleks metodu, linearno i nelinearno programiranje i dr.

Aproksimativne metode ne nalaze nužno optimum, već rješenje koje se nalazi blizu optimuma te su ga često sposobni pronaći u kratkom roku. U aproksimativne metode se ubrajaju heuristički algoritmi i aproksimacijski algoritmi. Aproksimacijski algoritmi ne nalaze nužno najbolje rješenje, ali daju formalno dokaziva jamstva izvedbe u obliku pronalaska rješenja koje je zadane bliskosti optimumu u zadano vrijeme izvršavanja algoritma. S druge strane, za heurističke algoritme najčešće ne postoje nikakve garancije bliskosti optimuma za rješenja koje pronalaze. Za neke razrede problema je moguće formalno dokazati da ih je teško rješavati aproksimacijskim algoritmima te su za te vrste problema heuristički postupci najčešće jedini način na koji ih je moguće rješavati.

Kako egzaktne metode nisu predmet ovog rada one više neće biti spominjane, nego će u daljnjem radu fokus biti na heurističkim metodama optimizacije.

2. HEURISTIČKE METODE OPTIMIZACIJE

Heuristika pronalazi dobro rješenje na velikom broju problemskih slučajeva. Ona dozvoljava postizanje prihvatljivog rješenja pri prihvatljivim troškovima u širokom rasponu problema. U općem smislu, heuristika nema garanciju aproksimacije na zadano rješenje. Heuristički algoritmi se mogu podijeliti u dvije podgrupe: specifična heuristika i metaheuristika. Specifična heuristika je usklađena i dizajnirana za rješavanje specifičnog problema i/ili slučajeva. Metaheuristika je algoritam generalne svrhe koja se može primijeniti na gotovo svaki optimizacijski problem. Može se sagledati kao generalna metodologija više razine koja se može koristiti kao strategija vođenja u dizajniranju temeljne heuristike za rješavanje specifičnih optimizacijskih problema.

Osim potrebe da se pronađe dobro rješenje za složenije probleme u razumnom vremenu, postoje i drugi razlozi za korištenjem heurističkih metoda, među kojima je bitno istaknuti sljedeće:

- heuristička metoda je više fleksibilnija od egzaktne metode i
- heuristička metoda koristi se kao dio globalnog postupka koji jamči pronalazak optimalnog rješenja problema.

Dobar heuristički algoritam mora ispuniti sljedeća svojstva [5]:

- rješenje se može dobiti razumnim računalnim trudom,
- rješenje treba biti blizu optimalnog (s velikom vjerojatnošću)
- te vjerojatnost za dobivanje lošeg rješenja bi trebala biti niska.

Uvjeti koje algoritam mora zadovoljiti kako bi imao navedena svojstva su:

- uvjet stabilnosti
- te uvjet oslobađanja iz lokalnog optimuma.

Ograničenja i nedostaci heuristike [6]:

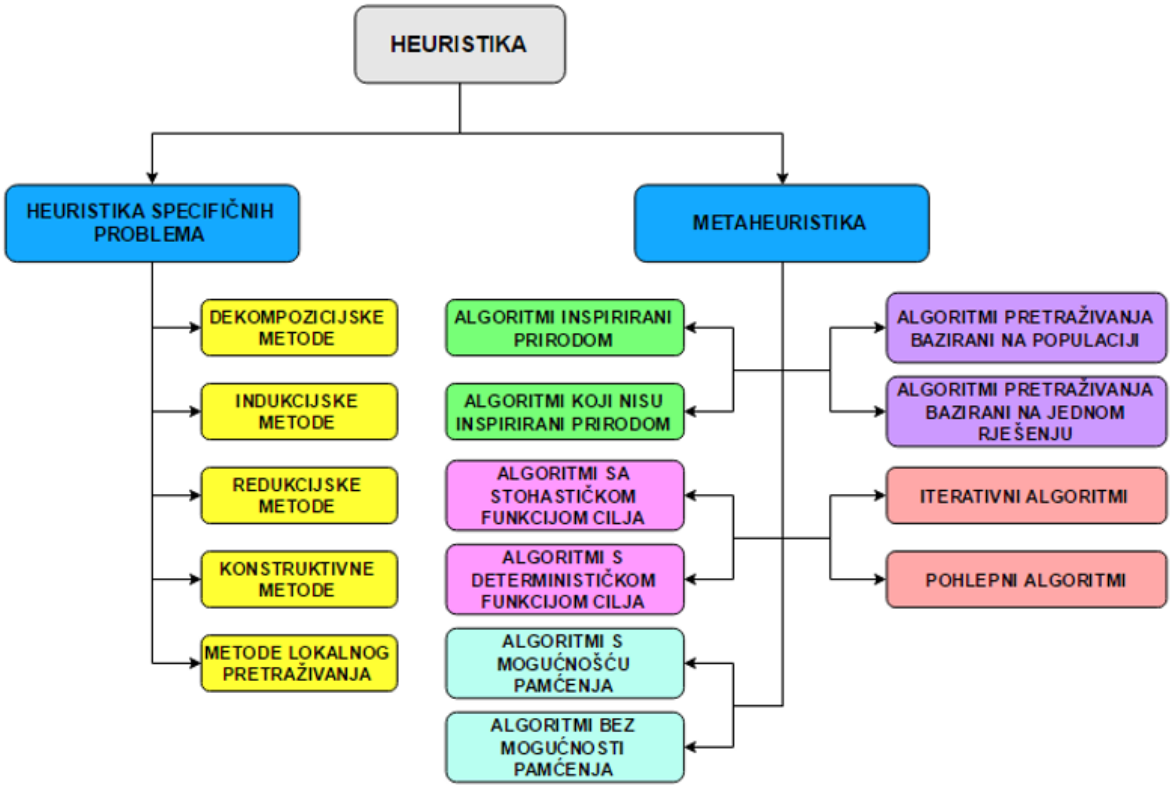
- najčešće rezultira suboptimalnim rješenjem, a ponekad i pronalaženjem bilo kakvog rješenja,
- koristi ograničene informacije poput trenutnog stanja rješenja,
- rijetko se može predvidjeti ponašanje algoritma u daljnjim koracima pretraživanja,
- te potrebna stručnost kod tumačenja rješenja.

2.1. Heuristika specifičnih problema

Postoji mnogo heurističkih metoda koje su vrlo različite u prirodi. Stoga je vrlo teško prikazati njihovu punu klasifikaciju. Osim toga, mnoge od njih su dizajnirane za rješavanje određenog problema, te se s ciljem pronalaska dobrih rješenja znatno oslanjaju na specifičnosti problema funkcije procjene kvalitete rješenja, bez mogućnosti generalizacije ili primjene na druge slične probleme. Zbog toga je izrada izravnog heurističkog postupka dugotrajan i naporan posao koji iziskuje detaljno istraživanje problema i duboko poznavanje njegovih karakteristika.

2.1.1. Podjela heuristike specifičnih problema

Stoga, kada se govori o metodama koje su dizajnirane za rješavanje određenog problema govori se o heuristici specifičnih problema, koja se dijeli na (Slika 2.1) dekompozicijske metode, indukcijske metode, redukcijske metode, konstruktivne metode i metode lokalnog pretraživanja.



Slika 2.1 Podjela heurističkih metoda [6, 7]

Dekompozicijske metode rade na principu razbijanja osnovnog problema u više potproblema.

Kod indukcijske metode, manje i jednostavnije verzije problema generalizacijom se primjenjuju na cijeli slučaj.

Redukcijske metode prepoznaju svojstva prihvatljivih rješenja, a ona rješenja koja ih ne posjeduju se eliminiraju. Cilj metode je ograničiti prostor mogućih rješenja pojednostavljenjem problema. Očiti rizik, kod ove metode, je mogući izostanak optimalnog rješenja originalnog problema.

Konstruktivne metode postepeno grade rješenje korak po korak, od nule. Obično se radi o determinističkim metodama koje imaju tendenciju odabira najboljeg rješenja u svakoj iteraciji.

Metode lokalnog pretraživanja, za razliku od ostalih metoda, polaze od proizvoljnog rješenja baziranog na iskustvu, koje se dalje poboljšava svakom sljedećom iteracijom.

Iako su sve ove metode pridonijele širenju znanja o rješavanju stvarnih problema, konstruktivne metode i metode lokalnog pretraživanja tvore temelje novim modernim heuristikama, takozvanoj metaheuristici.

2.2. Metaheuristika

Kako bi se smanjio trud za implementaciju heurističkih algoritama, nastao je niz novih heurističkih tehnika koje se nazivaju modernim heuristikama ili metaheuristikama. Za razliku od izravnih heuristika koje rješavaju jedan precizno definiran problem, metaheuristike su apstraktne i općenite strategije pretraživanja prostora rješenja koje se uz puno manji trud mogu prilagoditi za rješavanje konkretnog problema. Njihov rad se temelji na primjeni jednog ili oba principa:

- postupnoj izgradnji cjelokupnog rješenja iz pojedinačnih komponenti rješenja i
- modifikaciji postojećeg potpunog rješenja.

Metaheuristički postupci su široko primjenjivi razvojni okviri za izradu optimizacijskih postupaka. Njihova velika općenitost vrlo je povoljno svojstvo koje omogućuje njihovu primjenu na raznovrstan skup problema. U usporedbi s heuristikom specifičnog problema, veća općenitost omogućava njihovu bržu implementaciju. Posebno su prikladne kada je razumijevanje problema previše ograničeno za izradu izravne (specifične) heuristike ili nema dovoljno vremena za detaljna istraživanja.

Metaheuristika je doživjela veliku popularnost u posljednjih 20 godina. Upotreba metaheuristike u različitim granama industrije za rješavanje različitih i kompleksnih problema dokazala je njenu učinkovitost i djelotvornost. Neke od grana industrije u kojima se koristi metaheuristika su [3]:

- inženjerski dizajn, strukturalna optimizacija u elektrotehnici, aerodinamika, mehanika fluida, telekomunikacije, robotika i automobilska industrija,
- strojno učenje i rudarenje podataka u bioinformatici i računalnoj biologiji, te financijama,
- sustavno modeliranje, simuliranje i identificiranje u kemiji, fizici i biologiji,
- te planiranje ruta, robota, rasporeda, proizvodnje, logistike i transporta, lanca opskrbe, okruženja itd.

U izradi metaheuristike, dvije komponente moraju se uzeti u obzir: istraživanje prostora pretraživanja (diversifikacija) i eksploatacija najboljeg pronađenog rješenja (pojačavanje) [8]. Diversifikacija znači generiranje različitih rješenja, kako bi se istražio prostor pretraživanja na globalnoj razini, dok pojačavanje znači fokusiranje na pretraživanje lokalnog područja eksploatacijom informacija, da je trenutno dobro rješenje pronađeno u tom području. To je u kombinaciji s izborom najboljih rješenja. Izbor najboljeg osigurava konvergenciju rješenja s optimalnim rješenjem. S druge strane, diversifikacija povećava raznolikost rješenja, uz sprječavanje rješenja da budu uhvaćena na razini lokalnog optimuma. Sustav može biti zarobljen u lokalnom optimum ako ima premalo istraživanja ili previše eksploatacije. U tom slučaju, bilo bi vrlo teško ili čak nemoguće pronaći globalni optimum. S druge strane, ako ima previše istraživanja, a premalo eksploatacije, sustav će teško konvergirati. U tom slučaju, ukupna učinkovitost pretraživanja se usporava. Pronalaženje ravnoteže između tih dviju komponenti je, sam po sebi, problem optimizacije. Drugim riječima, eksploatacija i istraživanje su dio pretrage. Tijekom pretrage, treba uzeti u obzir odgovarajući mehanizam ili kriterij za odabir najboljeg rješenja. Čest kriterij za odabir najboljeg rješenja je „opstanak najjačih“. Kriterij se bazira na ponavljajućem ažuriranju trenutnog najboljeg rješenja koji je dosad pronađen. Svaki algoritam i njegove varijacije koriste različite načine za dobivanje ravnoteže između istraživanja i eksploatacije. Ukupna efikasnost algoritma ovisi o pronalaženju ravnoteže između tih dviju komponenta i obično to rezultira dobivanjem globalnog optimuma.

2.2.1. Podjela metaheuristike

Kako je prikazano na prethodnoj slici (Slika 2.1), metaheuristika se može klasificirati na više načina, a to su [3]:

- Algoritmi inspirirani prirodom nasuprot algoritama koji nisu inspirirani prirodom: Mnogi metaheuristički algoritmi su inspirirani prirodnim procesima: evolucijskim

algoritmima i umjetnim imunološkim sustavima iz biologije; kolonije mrava, kolonije pčela, optimizacija rojem čestica iz inteligencije roja različitih vrsta (društvene znanosti), te simulirano kaljenje iz fizike.

- Algoritmi sa stohastičkom funkcijom cilja nasuprot algoritama s determinističkom funkcijom cilja: Deterministička metaheuristika rješava problem optimizacije donošenjem determinističkih odluka (npr. algoritam lokalnog pretraživanja, tabu pretraživanja). U stohastičkoj metaheuristici, neka slučajna pravila se primjenjuju tijekom pretraživanja (npr. simulirano kaljenje, evolucijski algoritmi). Kod determinističkih algoritama, upotreba istog početnog rješenja dovodi do istog konačnog rješenja, dok kod stohastičkih algoritama, različita konačna rješenja su dobivena iz istog početnog rješenja. Ova karakteristika mora se uzeti u obzir kod izvođenja evolucije metaheurističkog algoritma.
- Algoritmi s mogućnošću pamćenja nasuprot algoritama bez mogućnosti pamćenja: Neki metaheuristički algoritmi su bezmemorijski, odnosno, ne koriste informacije koje su dobivene dinamički, tijekom pretraživanja. Neki predstavnici ove klase su: algoritam lokalnog pretraživanja, GRASP metoda te simulirano kaljenje. Ostale metaheuristike koriste memoriju koja sadrži neke informacije koje su izvađene tijekom pretraživanja. Na primjer, kratkoročna i dugoročna memorija kod tabu pretraživanja.
- Algoritmi pretraživanja bazirani na populaciji nasuprot algoritama pretraživanja baziranih na jednom rješenju: Algoritam baziran na jednom rješenju (npr. algoritam lokalnog pretraživanja, simulirano kaljenje) upravlja i transformira jedno rješenje tijekom pretraživanja, dok kod algoritama baziranih na populaciji (npr. roj čestica, evolucijski algoritmi) cijela populacija rješenja je uključena. Ove dvije klase metoda imaju komplementarne karakteristike. Algoritmi bazirani na jednom rješenju su eksploatacijski orijentirani te imaju snagu intenzivnog pretraživanja u lokalnom području, a algoritmi bazirani na populaciji su orijentirani istraživanju i omogućavaju bolju diversifikaciju u cijelom prostoru istraživanja. Algoritmi koji pripadaju ovim dvjema kasama dijele mnoge mehanizme pretraživanja.
- Iterativni algoritmi nasuprot pohlepnim algoritmima: U iterativnim algoritmima kreće se od potpunog rješenja (ili od populacije rješenja), te ga se u svakoj sljedećoj iteraciji transformira koristeći neke operatore pretraživanja. Za razliku od iterativnog, pohlepni algoritam započinje svoj rad bez rješenja i pri svakom koraku dodjeljuje se varijabla

odluke problema dok se ne dobije kompletno rješenje. Većina metaheuristike su iteracijski algoritmi.

Također, prema autoru Dan Stefanou metaheuristika se može klasificirati prema pristupu na lokalne i globalne metode [9]. U lokalne metode ubrajaju se: metoda Monte Carlo, *Hill Climbing* metoda, metoda tabu pretraživanja, metoda simuliranog kaljenja, GRASP metoda i dr. U globalne metode ubrajaju se: genetički algoritmi, optimizacija kolonijom mrava, rojem čestica, rojem pčela i dr.

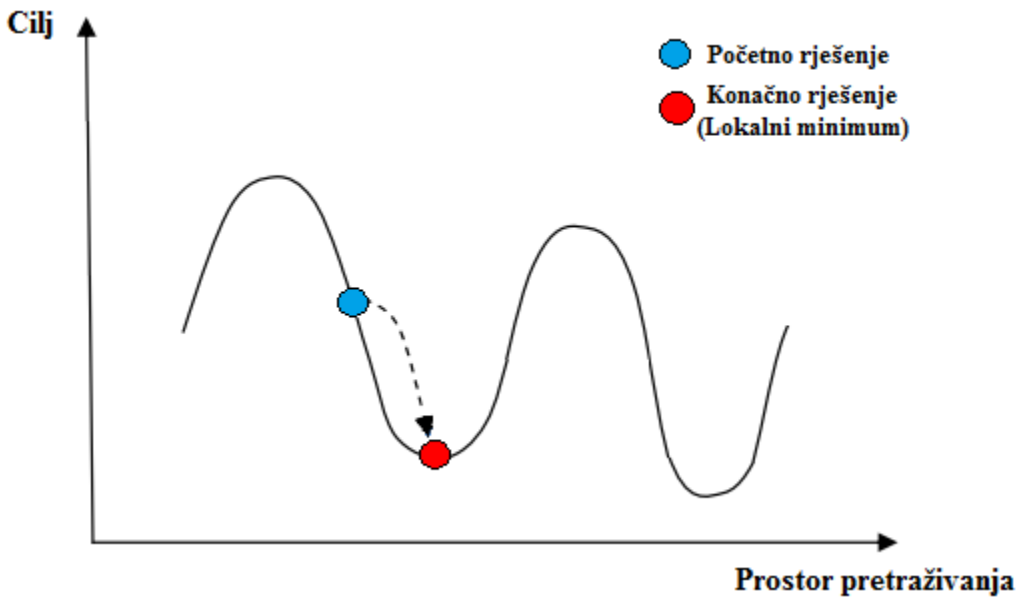
3. METAHEURISTIKA - LOKALNE METODE

S točke pristupa, algoritmi se mogu klasificirati kao lokalni i globalni. Lokalni algoritmi pretraživanja obično konvergiraju prema lokalnom optimumu, koji nije nužno i globalni, i takvi algoritmi često su deterministički te nemaju sposobnost bijega iz lokalnog optimuma. No, ima i onih koji imaju sposobnost bijega, kao što su metoda simuliranog kaljenja i metoda tabu pretraživanja.

U ovome poglavlju opisat će se nekoliko metaheuristika baziranih na lokalnom pristupu, a to su: metoda lokalne pretrage, metoda simuliranog kaljenja, metoda tabu pretraživanja, GRASP metoda te metoda Monte Carlo.

3.1. Metoda lokalnog pretraživanja

Lokalno pretraživanje (eng. LS - *Local Search*) je najstarija i najjednostavnija metaheuristička metoda [3]. Metoda kreće s danim početnim rješenjem. Pri svakoj iteraciji, heuristika zamjenjuje postojeće rješenje s boljim susjednim rješenjem. Potraga staje kada su sva susjedna rješenja gora od trenutnog rješenja. Što znači da je postignut lokalni optimum (Slika 3.1).



Slika 3.1 Ponašanje LS algoritma u danom okruženju [3]

LS se može sagledati kao šetnja prema dolje u grafu koji predstavlja prostor pretrage. Ovaj se graf može opisati kao $G = f(S, V)$, gdje S predstavlja skup svih mogućih rješenja u prostoru pretraživanja, a V vezu između rješenja. U nastavku je dan algoritam lokalnog pretraživanja.

Algoritam 3.1 Algoritam lokalnog pretraživanja [3]

s = ***s*₀ ;** /* Stvaranje polaznog rješenja s_0 */
Dok se ne postigne kriterij zaustavljanja **Čini**
    Generiraj [$N(s)$] ; /* Generiranje susjednih rješenja */
    Ako nema boljeg susjednog rješenja **Onda** stani ;
    s = ***s'*** ; /* Odaberi bolje susjedno rješenje $s' \in N(s)$ */
Ispiši najbolje pronađeno rješenje (lokalni optimum)

3.1.1. Odabir susjednog rješenja

Kod odabira susjednog rješenja može se primijeniti jedna od sljedeće tri strategije [3]:

- Najveće poboljšanje: U ovoj strategiji, odabire se susjedno rješenje koje najviše poboljšava trenutno rješenje.
- Prvo poboljšanje: Ova strategija se temelji na odabiru prvog poboljšavajućeg rješenja, tj. prvog pronađenog susjednog rješenja koje uzrokuje poboljšanje trenutnog rješenja.
- Slučajni odabir: Kod ove strategije, slučajni odabir se primjenjuje na ona susjedna rješenja koja mogu poboljšati postojeće rješenje.

Kompromis u pogledu kvalitete rješenja i vremena pretraživanja može se sastojati u upotrebi strategije prvog poboljšanja kada je početno rješenje nasumično generirano, a strategija najvećeg poboljšanja kada je početno rješenje generirano pomoću pohlepnog postupka. Praksa je, u mnogim primjenama, pokazala da strategija prvog poboljšanja dovodi do iste kvalitete rješenja kao i strategija najvećeg poboljšanja, ali u kraćem vremenskom periodu. Osim toga, vjerojatnost prerane konvergencije na lokalni optimum je manje važna kod strategije prvog poboljšanja.

3.1.2. Izlazak iz lokalnog optimuma

U principu, lokalno pretraživanje je vrlo jednostavna metoda za izradu i provedbu te daje vrlo brzo prilično dobra rješenja. To je razlog zašto metoda ima široku primjenu u praksi. Jedan od glavnih nedostataka LS metode je da konvergira prema lokalnom optimumu. Osim toga, algoritam može biti jako osjetljiv na početno rješenje, tj. može se dobiti velika varijacija kvalitete rješenja za neke probleme. Osim toga, ne postoji način za procjenu relativne pogreške u odnosu na globalni optimum i broj potrebnih iteracija se ne može znati unaprijed. Najgori slučaj složenosti LS metode je eksponencijalan. LS radi dobro ako nema previše lokalnih optimuma u prostoru pretraživanja ili kvaliteta drugog lokalnog optimuma je manje ili više jednaka.

Kako je glavni nedostatak LS algoritma konvergencija prema lokalnom optimumu, mnogi alternativni algoritmi su predloženi kako bi se izbjeglo zaustavljanje u lokalnom optimumu. Razlikujemo četiri različita pristupa [3]:

1. Provođenje iteracija s više početnih rješenja: Ova strategija se primjenjuje kod iterativnog pretraživanja, GRASP itd.
2. Prihvaćanje ne-poboljšavajućih susjednih rješenja: Ova strategija omogućava pristupe koji degradiraju trenutno rješenje, tj. omogućava da se pretraga preseli iz određenog lokalnog optimuma. Metoda simuliranog kaljenja i metoda tabu pretraživanja su popularni predstavnici ovog pristupa.
3. Promjena rješenja: Ovaj pristup sastoji se od promjene strukture rješenja tijekom pretraživanja.
4. Promjena funkcije cilja ili ulaznih podataka o problemu: U ovom pristupu, problem je preobražen mijenjanjem ulaznih podataka, funkcije cilja ili ograničenja, u nadi da će se riješiti učinkovitije izvorni problem optimizacije.

3.2. Metoda simuliranog kaljenja

Simulirano kaljenje (eng. SA - *Simulated Annealing*) je metoda bazirana na statističkoj mehanici, gdje postupak kaljenja zahtjeva zagrijavanje pa zatim sporo hlađenje tvari, kako bi se dobila snažna kristalna struktura [3]. Čvrstoća konstrukcije ovisi o brzini hlađenja metala. Ako polazna temperatura hlađenja nije dovoljno visoka ili ako se provodi prebrzo hlađenje, dobivena je nesavršenost (metastabilno stanje). U tom slučaju, hlađenjem se neće postići toplinska ravnoteža na svakoj temperaturi. Snažni kristali nastaju iz pažljivog i sporog hlađenja. SA algoritam simulira promjene energije u sustavu podvrgnutom procesu hlađenja dok ne konvergira u ravnotežno stanje.

Tablica 3.1 prikazuje analogiju između fizikalnog sustava i problema optimizacije. Funkcija cilja analogna je energetskom stanju sustava. Rješenje problema optimizacije odgovara stanju sustava. Varijable odluke povezane s rješenjem problema optimizacije analogne su pozicijama molekula. Globalni optimum odgovara osnovnom stanju sustava, a pronalazak lokalnog minimuma sugerira da je postignuto metastabilno stanje.

Tablica 3.1 Analogija između fizikalnog sustava i problema optimizacije

Fizikalni sustav	Problem optimizacije
Stanje sustava	**Rješenje**
Pozicija molekula	**Varijable odluke**
Energija	**Funkcija cilja**
Osnovno stanje	**Globalni optimum**
Metastabilno stanje	**Lokalni optimum**
Brzo gašenje	**Lokalno pretraživanje**
Temperatura	**Parametar kontrole**
Kaljenje	**Simulirano kaljenje**

SA je stohastički algoritam koji omogućuje pod nekim uvjetima degradaciju rješenja. Cilj algoritma je izlazak iz lokalnog optimuma te odgoda konvergencije. SA je algoritam bez pamćenja, u smislu da algoritam ne koristi informacije prikupljene tijekom pretraživanja. Od početnog rješenja, SA algoritam se odvija u nekoliko ponavljanja. U svakoj iteraciji, generirano je susjedno rješenje. Potezi koji poboljšavaju funkciju cilja su uvijek prihvaćeni. Susjedno rješenje je odabrano s vjerojatnošću koja ovisi o trenutnoj temperaturi i količini degradacije ΔE funkcije cilja. ΔE predstavlja razliku u vrijednosti funkcije cilja (energije) između trenutnog rješenja i generiranog susjednog rješenja. Kako algoritam napreduje, vjerojatnost da će takvi potezi biti prihvaćeni opada. Ova vjerojatnost, u cijelosti, prati Boltzmannovu distribuciju [3]:

$$P(\Delta E, T) = \mathrm{e}^{-\frac{f(s') - f(s)}{T}} \tag{1}$$

Algoritam koristi parametar kontrole, zvan temperatura, kako bi se utvrdila vjerojatnost prihvaćanja rješenja koje ne poboljšava trenutno rješenje. Kada se postigne ravnotežno stanje temperatura se postupno smanjuje prema rasporedu za hlađenje, tako da je na kraju istraživanja malo prihvaćenih rješenja koja ne poboljšavaju prethodno rješenje. U daljnjem tekstu dan je predložak algoritma simulacijskog kaljenja (Algoritam 3.2):

Algoritam 3.2 Algoritam simuliranog kaljenja [3]

Unesi raspored hlađenja
s **=** s_0 **;** /* Stvaranje polaznog rješenja */
$T = T_{max}$ **;** /* Polazna temperatura */
Ponavljaj
    Ponavljaj /* Na određenoj temperaturi */
        Generiraj slučajno susjedno rješenje *s'* ;
        $\Delta E = f(s') - f(s)$ **;**
        Ako $\Delta E \leq 0$ **Onda** ***s*** **=** ***s'*** /* Prihvati susjedno rješenje */

Inače Prihvati s' s vjerojatnošću $e^{\frac{-\Delta E}{T}}$
Sve dok uvjet ravnoteže nije zadovoljen
/* npr. zadani broj iteracija za izvršiti na svakoj temperaturi *T* */
***T* = *g*(*T*) ;** /* Ažuriraj temperaturu */
Sve dok se ne zadovolji kriterij zaustavljanja/* npr. $T < T_{min}$ */
Ispiši najbolje pronađeno rješenje

U nastavku ovog poglavlja detaljnije su objašnjeni neki od korištenih parametara u algoritmu simulacije kaljenja.

3.2.1. Prihvaćanje poteza

Funkcija vjerojatnosti prihvaćanja je glavni element SA algoritma koja omogućuje da susjedna rješenja koja ne uzrokuju poboljšanje budu prihvaćena. Sustav može pobjeći od lokalnog optimuma zbog vjerojatnosti prihvaćanja susjednog rješenja koje ne poboljšava postojeće rješenje. Vjerojatnost prihvaćanja susjednog rješenja, koje ne uzrokuje poboljšanje, proporcionalna je temperaturi T i obrnuto proporcionalna promjeni funkcije cilja ΔE. Zakon termodinamike kaže da je na temperaturi T vjerojatnost povećanja energije E jednak [3]:

$$P(\Delta E, T) = e^{-\frac{\Delta E}{k \cdot T}} > R \tag{2}$$

gdje je ΔE promjena u evaluacijskoj funkciji, T trenutna temperatura, k Boltzmannova konstanta i R je uniformni slučajni broj između 0 i 1. Pri visokim temperaturama vjerojatnost prihvaćanja lošeg poteza je veća. Ako je $T = \infty$, svi potezi su prihvaćeni, što odgovara slučajnom lokalnom hodu u okruženju. Kod nižih temperatura, vjerojatnost prihvaćanja lošeg poteza je niža. Ako je $T = 0$, prihvaćanje lošijeg poteza nije prihvaćeno i potraga je ekvivalentna lokalnoj potrazi (npr. *Hill Climbing*). Osim toga, vjerojatnost prihvaćanja velikih pogoršanja u kvaliteti rješenja opada eksponencijalno prema 0 prema Boltzmannovoj razdiobi.

3.2.2. Raspored hlađenja

Raspored hlađenja određuje za svaki korak algoritma temperaturu T_i. Ima veliki utjecaj na uspjeh SA algoritma. Performanse SA algoritma su jako osjetljive na izbor rasporeda za hlađenje. Parametri koji se uzimaju u obzir prilikom određivanja rasporeda hlađenja su polazna temperatura, ravnotežno stanje, funkcija hlađenja i krajnja temperatura koja definira kriterij zaustavljanja.

3.2.2.1. Polazna temperatura

Ako je polazna temperatura vrlo visoka, pretraživanje će biti manje ili više jednako lokalnom pretraživanju. Inače, ako je polazna temperatura vrlo niska, pretraživanje će biti manje ili više jednako prvom poboljšanju algoritma za lokalno pretraživanje. Dakle, mora se uspostaviti ravnoteža između tih dviju ekstremnih situacija. Polazna temperatura ne smije biti previsoka za obavljanje slučajnog pretraživanja za određeno vremensko razdoblje, ali dovoljno visoka kako bi omogućila potez.

Postoje tri glavne strategije koje se mogu koristiti za rješavanje ovog parametra [3]:

1. Prihvatit sve: Polazna temperatura je postavljena dovoljno visoko da prihvati sva susjedna rješenja u početnoj fazi algoritma. Glavni nedostatak ove strategije je njeno dugo vrijeme računanja.
2. Prihvaćanje odstupanja: Polazna temperatura se izračunava s $k \cdot \sigma$ preliminarnih pokusa, gdje σ predstavlja standardno odstupanje između vrijednosti funkcije cilja, a $k = \frac{-3}{\ln(p)}$ s vjerojatnošću prihvaćanja p, koja je viša od 3σ (99,73 %) [10].
3. Prihvaćanje omjera: Polazna temperatura je definirana tako da omjer prihvaćanja rješenja bude veći od prethodno određene vrijednosti a_0:

$$T_0 = \frac{\Delta^+}{\ln[\frac{m_1 \cdot (a_0 - 1)}{m_2} + a_0]} \tag{3}$$

gdje su m_1 i m_2 brojevi rješenja koja se povećavaju ili smanjuju u preliminarnim eksperimentima, a Δ^+ je prosjek povećanja funkcije cilja. Na primjer, polaznu temperaturu treba odrediti na takav način da je omjer prihvaćanja u intervalu [0,4; 0,5].

3.2.2.2. Ravnotežno stanje

Da bi se postiglo ravnotežno stanje pri svakoj temperaturi, mora se primijeniti niz prijelaza. Teorija sugerira da broj iteracija pri svakoj temperaturi može biti eksponencijalan s veličinom problema, što je komplicirana strategija za primjenu u praksi. Broj iteracija mora biti postavljen u skladu s veličinom problema i posebno proporcionalan veličini susjedstva $N(s)$. Broj potrebnih prijelaza se može odrediti [3]:

- Statički: U ovoj strategiji broj prijelaza se određuje prije početka potrage.
- Prilagodljivi: Broj generiranih susjednih rješenja ovisi o karakteristikama pretrage. Npr. nije potrebno da se dosegne ravnotežno stanje pri svakoj temperaturi. Mogu se koristiti algoritmi neravnotežnog simuliranja kaljenja. Kod njih, raspored hlađenja se može

izvršiti odmah nakon što je generirano poboljšavajuće susjedno rješenje. Ovo rezultira smanjenjem vremena računanja bez ugrožavanja kvalitete dobivenih rješenja.

3.2.2.3. Hlađenje

Uvijek postoji kompromis između kvalitete dobivenih rješenja i brzine hlađenja. Ako se temperatura polako smanjuje rješenja su bolja, ali sa značajnijim vremenom računanja. Temperatura T se može ažurirati na različite načine [3]:

- Linearno: U trivijalnom linearnom rasporedu, temperatura T je ažurirana prema:
  $$T = T - \beta \tag{4}$$
  gdje je β specifična konstanta vrijednost. Stoga, $T_i = T_0 - i \cdot \beta$ predstavlja temperaturu T pri i-toj iteraciji.
- Geometrijski: U geometrijskom rasporedu, temperatura je ažurirana prema:
  $$T = \alpha \cdot T \tag{5}$$
  gdje je $\alpha \in [0, 1]$. To je ujedno i najpopularnija funkcija hlađenja. Iskustvo je pokazalo da bi α trebala biti između 0,5 i 0,99.
- Logaritamski: Koristi se sljedeća formula:
  $$T = \frac{T_i}{\log(i)} \tag{6}$$
  Ovaj raspored je prespor za primjenu u praksi, ali ima svojstvo konvergencije do globalnog optimuma.
- Vrlo sporo hlađenje: Kod ovog načina se provodi veliki broj iteracija na nekoliko razina temperature ili mali broj iteracija na mnogo razina temperature. Veoma spori raspored hlađenja određuje se prema:
  $$T_{i+1} = \frac{T_i}{1 + \beta \cdot T_i} \tag{7}$$
  gdje je $\beta = T_0 - T_F / (L - 1) \cdot T_0 \cdot T_F$, T_F konačna temperatura i L sljedeći broj prijelaza. Samo jedna iteracija je dozvoljena pri svakoj temperaturi u ovoj sporopadajućoj funkciji.
- Prilagodljivo: Većina rasporeda hlađenja je statična u smislu da se raspored hlađenja određuje potpuno *a priori*. U tom slučaju raspored hlađenja je slijep na karakteristike terena koji se pretražuje. U tom rasporedu hlađenja, brzina hlađenja je dinamična i ovisi o informacijama prikupljenih u toku pretraživanja. Dinamičan raspored hlađenja može se koristit kada se provodi mali broj ponavljanja pri visokim temperaturama i veliki broj ponavljanja pri niskim temperaturama.

3.2.2.4. Kriterij zaustavljanja

U pogledu kriterija zaustavljanja, teorija sugerira konačnu temperaturu jednaku 0. U praksi, može se zaustaviti potraga kada je vjerojatnost prihvaćanja poteza zanemariva. Kriteriji zaustavljanja koji postoje [3]:

- Postizanje konačne temperature T_F je najpopularniji kriterij zaustavljanja. Ova temperatura mora biti vrlo niska (npr. $T_{min} = 0{,}01$).
- Postizanje unaprijed određenog broja iteracija bez unapređenja najboljeg pronađenog rješenja.
- Postizanje unaprijed određenog broja puta prihvaćanja susjednog rješenja pri svakoj temperaturi, tj. brojač se povećava za 1, svaki put kad je temperatura završena s manjim postotkom od unaprijed određene granice, i resetira se na 0 kada se pronađe novo najbolje rješenje. Ako brojač dosegne određenu granicu *R*, SA algoritam se zaustavlja.

3.3. Tabu pretraživanje

Algoritam tabu pretraživanja (eng. TS - *Tabu Search*) predložio je Fred Glover 1986. godine [11]. TS je stohastička metoda globalne optimizacije, koja je originalno razvijena za rješavanje veoma velikih kombinatornih problema optimizacije te je kasnije proširena na kontinuiranu optimizaciju. Kao SA, TS je algoritam pretraživanja baziran na jednom rješenju. TS algoritam koristi niz strategija i naučenih informacija za oponašanjem ljudske spoznaje za rješavanjem problema. TS je uveo sustavno istraživanje memorije u procesima pretraživanja, dok su evolucijski algoritmi uveli ideju kombiniranog rješenja. TS algoritam je konceptno jednostavniji od SA algoritma ili evolucijskih algoritama, te je jednostavan za implementaciju. Također, TS algoritam superioran je prethodno navedenim algoritmima u mnogim optimizacijskim problemima, kako po pitanju vremena računanja tako i po kvaliteti rješenja.

TS je iterativni postupak koji polazi od danog početnog rješenja i pokušava ga unaprijediti u svakoj iteraciji. Osnovni princip tabu pretraživanja je da nastavi s pretragom kad god naiđe na lokalni optimum prihvaćanjem poteza koji ne donosi poboljšanje (ako nijedno susjedno rješenje ne donosi poboljšavanje rješenja) s ciljem da se lokalni optimum savlada. Kako se pretraga ne bi vraćala (stvarali ciklusi) u već posjećena rješenja potezi koji vode do njih postaju zabranjeni, tj. postaju tabu potezi. Informacije o pretrazi se čuvaju u memoriji, koja se ažurira nakon svake iteracije, a vraćanje u već posjećena rješenja spriječeno je na osnovi tih informacija. Ključna ideja je da se za upravljanje pretraživanja koriste informacije o prethodnim

pretraživanjima. To se može povezati s metodama informacijskog pretraživanja u polju umjetne inteligencije. U nastavku je dan predložak algoritma tabu pretraživanja (Algoritam 3.3):

Algoritam 3.3 Algoritam tabu pretraživanja [3]

s = ***s*₀ ;** /* Stvaranje polaznog rješenja */
Polazna tabu lista, srednjoročna i dugoročna memorija ;
Ponavljaj
    Pronađi najbolje dopustivo susjedno rješenje *s*' ;
    s = ***s*'** ;
    Ažuriraj tabu listu, kriterij aspiracije, srednjoročnu i dugoročnu memoriju ;
    Ako kriterij pojačavanja postoji **Onda** pojačavaj ;
    Ako kriterij diversifikacije postoji **Onda** diversificiraj ;
Sve dok se ne postigne kriterij zaustavljanja
Ispiši najbolje pronađeno rješenje

3.3.1. Memorija TS algoritma

Kako je prikazano u donjoj tablici (Tablica 3.2) memorija TS algoritma se može podijeliti na tabu listu, srednjoročnu memoriju i dugoročnu memoriju.

Tablica 3.2 Memorije TS algoritma

Memorija TS	Uloga
Tabu lista	**Spriječiti ciklus**
Srednjoročna memorija	**Pojačavanje**
Dugoročna memorija	**Diversifikacija**

Uloga tabu liste je spriječiti ciklus ponavljanja, srednjoročne memorije eksploatacija informacija radi navođenja pretrage u obećavajućem području, a dugoročne memorije poticanje diversifikacije pretraživanja.

3.3.1.1. Tabu lista

Riječ tabu, ili *taboo*, dolazi s otoka Togan, jednog od otoka Polinezije [12], gdje ga koriste domoroci za stvari koje se ne smiju dirati jer su svete. Stoga, tabu u slobodnom prijevodu znači zabranjen ili zabranjeno. Kao što je već spomenuto, tabui se koriste kako bi se spriječili ciklusi prilikom udaljavanja od lokalnog optimuma kroz poteze koji ne dovode do boljih rješenja. Kada proces pretrage dospije u lokalni optimum, izlazak iz njega se pokušava potezima koji ne dovode do poboljšanja rješenja. Tim potezima se u okolini trenutnog rješenja bira novo rješenje, koje nije bolje od trenutnog, i proces pretrage se nastavlja u okolini tog novog rješenja.

Neophodno je onemogućiti da se pretraga vrati u već posjećena rješenja jer time nastaju nepotrebni ciklusi, a to se postiže tako da se posjećena rješenja proglase zabranjenim, tj. tabuima, u određenom broju sljedećih iteracija. Potpuna rješenja bi se mogla čuvati, ali to nije efikasno jer zahtjeva veliku količinu memorije. Uz to, potrebno je dosta vremena za provjeru da li je potencijalno rješenje tabu ili ne, pa se ova opcija rijetko koristi. Alternativa je da se umjesto cijelih rješenja zabrane potezi koji vode do njih. Kako će se u tom slučaju predstaviti tabui zavisi od karakteristike problema i načina predstavljanja rješenja.

Veličina tabu liste (popisa) je kritični parametar koji ima veliki utjecaj na algoritam tabu pretrage. Pri svakoj iteraciji, zadnji potez je dodan na tabu listu, dok je najstariji uklonjen. Što je manja vrijednost tabu liste, veća je vjerojatnost pojave ciklusa. Veće vrijednosti tabu liste omogućit će mnoga ograničenja i poticanje diversifikacije. Mora se uspostaviti kompromis između tih dviju vrijednosti. Tabu lista može poprimiti različite oblike [3]:

- Statički: U principu, statička vrijednost je povezana s tabu listom. To može ovisiti o veličini problema, a posebice o veličini susjedstva. Ne postoji optimalna vrijednost tabu liste za sve probleme ili za sve slučajeve određenog problema. Zapravo, optimalna vrijednost može varirati tijekom pretraživanja. Da bi se prevladao problem, može se koristit tabu lista s različitim veličinama.
- Dinamički: Veličina tabu liste može se promijeniti tijekom pretraživanja bez korištenja bilo kakve informacije iz memorije pretraživanja.
- Prilagodljivi: Kod adaptivnog oblika, veličina tabu liste se ažurira u skladu s memorijom pretraživanja. Npr. veličina liste je ažurirana nakon obavljene pretrage u posljednjoj iteraciji.

3.3.1.2. Srednjoročna memorija

Uloga pojačavanja je eksploatacija informacija najboljih pronađenih rješenja (elitnih rješenja) radi navođenja potrage u obećavajućoj regiji područja pretraživanja [3]. Ova informacija je sadržana u srednjoročnoj memoriji. Ideja se sastoji u vađenju zajedničkih obilježja elitnih rješenja, te zatim pojačavanja potrage oko rješenja koja dijele ta obilježja. Popularni pristup se sastoji od ponovnog pokretanja potrage s najboljim rješenjem i zatim fiksiranja u tom rješenju najperspektivniju komponentu izvađenu iz elitnih rješenja.

Glavni prikaz koji se koristi za srednjoročnu memoriju je nedavna memorija. Najprije treba definirati komponente povezane s rješenjem. Nedavna memorija će za svaku komponentu u posjećenom rješenju zapamtiti broj iteracija. Najčešće korišteni kriterij za pokretanje procesa

pojačavanja je dano razdoblje ili određeni broj iteracija bez poboljšanja. Pojačavanje pretrage u određenoj regiji prostora pretraživanja nije uvijek korisno. Učinkovitost pojačanja ovisi o strukturi problema optimizacije.

3.3.1.3. Dugoročna memorija

Dugoročna memorija je uvedena u TS metodu radi poticanja diversifikacije pretrage. Uloga dugoročne memorije je prisiliti potragu u neistraženim regijama prostora pretraživanja. Glavni prikaz dugoročne memorije je frekvencijska memorija. Kao i u nedavnoj memoriji, komponente povezane s rješenjem moraju biti definirane. Frekvencijska memorija će zapamtiti koliko je puta pojedina komponenta bila prisutna u svim posjećenim rješenjima. Proces diversifikacije može biti primijenjen periodički ili nakon određenog broja iteracija bez poboljšanja. Kao i kod pojačavanja, diversifikacija pretraživanja nije uvijek korisna. To ovisi o strukturi problema optimizacije. Mogu se primijeniti tri popularne strategije [3]:

- Ponovno pokretanje diversifikacije: Ova strategija se sastoji od uvođenja u trenutno ili najbolje rješenje najmanje posjećene komponente. Nakon toga, potraga se ponavlja od tog novog rješenja.
- Kontinuirana diversifikacija: Ova strategija predstavlja, za vrijeme pretraživanja, pristranost poticanja diversifikacije. Na primjer, funkcija cilja može integrirati pojavu učestalosti komponenti rješenja u procjeni postojećih rješenja. Često primjenjivani potezi ili posjećivana rješenja se kažnjavaju.
- Strateška oscilacija: Strateška oscilacija omogućuje razmatranje osrednjih rješenja koja su nemoguća. Ova strategija navodi pretragu prema nemogućim rješenjima, a zatim je vraća prema mogućim.

3.3.2. Kriterij aspiracije

I kada ne postoji mogućnost nastanka ciklusa, neki jako dobri potezi mogu biti zabranjeni što može dovesti do potpune stagnacije procesa pretraživanja. Stoga je neophodno uvesti mehanizam koji će omogućiti opoziv tabu statusa nekom potezu. Taj mehanizam naziva se kriterij aspiracije [13]. Najjednostavniji i najčešće korišteni kriterij aspiracije, koji se sreće u gotovo svim implementacijama tabu pretraživanja, sastoji se od odabira tabu poteza koji generira bolje rješenje od pronađenog najboljeg. Drugi kriterij aspiracije može biti tabu potez koji daje bolje rješenje, unutar skupa rješenja, koji posjeduje određeni atribut.

3.3.3. Kriterij zaustavljanja

Najčešće korišteni kriteriji zaustavljanja koji se koriste su: zaustaviti algoritam ako je broj uzastopnih iteracija bez poboljšanja funkcije cilja veći od neke vrijednosti ili ako je postignuta točna i procijenjena optimalna vrijednost funkcije cilja.

3.4. GRASP metoda

GRASP metoda (eng. GRASP - *Greedy Randomized Adaptive Search Procedure*) je iterativna pohlepna heuristika za rješavanje kombinatorne optimizacije [14]. Svaka iteracija GRASP algoritma sastoji se od dva koraka: koraka izgradnje i koraka lokalnog pretraživanja. U koraku izgradnje, moguće rješenje je izgrađeno korištenjem nasumičnog pohlepnog algoritma, dok se u drugom koraku heuristika lokalnog pretraživanja primjenjuje na izgrađeno rješenje s ciljem poboljšanja rješenja pronađenog u koraku izgradnje. Pohlepni algoritam mora biti nasumičan kako bi bio u mogućnosti generirati različita rješenja. Inače, postupak lokalnog pretraživanja može biti primijenjen samo jednom. Ovaj postupak od dva koraka se ponavlja sve dok zadani broj iteracija i najbolje pronađeno rješenje nije uzeto kao konačno. Pošto su iteracije potpuno neovisne, nema potrebe za memorijom pretraživanja. Algoritam 3.4 prikazuje predložak za GRASP algoritam:

Algoritam 3.4 Algoritam GRASP-a [3]

Unesi broj iteracija
Ponavljaj
    s = *Pohlepna_heuristika* ; /* Primijeniti nasumičnu pohlepnu heuristiku */
    s' = *Lokalno_pretraživanje*(*s*) ; /* Primijeniti LS algoritam na rješenje */
Sve dok se ne zadovolji kriterij zaustavljanja /* Zadan broj iteracija */
Ispiši najbolje pronađeno rješenje

Glavna projektna pitanja za GRASP metodu su pohlepna izgradnja i postupci lokalnog pretraživanja.

3.4.1. Korak izgradnje

Pohlepni algoritam uvijek odabire izbor koji izgleda najbolje u danom trenutku. To može dovesti lokalni optimalni izbor do globalnog optimuma. Pohlepni algoritam ne daje uvijek optimalna rješenja, ali u nekim slučajevima to čini (npr. Najmanje razapinjuće stablo) [14]. Problem najmanjeg razapinjućeg stabla (eng. MST - *Minimum Spanning Tree*) je ključni problem u dizajnu mreže, kao što su transportne i telekomunikacijske mreže. Stablo se sastoji od povezivanja niza čvorova na korijen čvorova kroz minimalne troškove uz poštivanje danih

ograničenja. Takvim povezivanjem dobiva se povezani neusmjereni graf $G = f(V, E)$, gdje $V = \{v_1, ... v_n\}$ predstavlja skup vrhova, a E skup bridova. Svaki čvor, odnosno rub, je ponderiran pozitivnom cijelom broju b_i (odnosno $c_{i,j}$). Teret $c_{i,j}$ predstavlja trošak ruba, dok b_i predstavlja protok čvora. Broj Q označava sposobnost koja ne smije biti prekoračena od strane bilo kojeg ruba, a čvor $r \in V$ definira ponorište. MST se sastoji od pronalaženja minimalnog razapinjućeg stabla, koji minimizira ukupne troškove rubova i zadovoljava sljedeće ograničenje: suma težina vrhova u svakoj komponenti grafa je manja ili jednaka Q.

U konstruktivnoj heuristici, u svakoj iteraciji elementi koji se mogu uključiti u djelomično rješenje su poredani u listu korištenjem lokalne heuristike. Pomoću ove liste, generiran je podskup koji predstavlja ograničeni popis kandidata (eng. RCL - *Restricted Candidate List*). RCL lista je ključna komponenta GRASP metaheuristike. Ona predstavlja aspekt vjerojatnosti metaheuristike. Kriterij ograničenja može ovisiti o [3]:

- Kardinalno baziranom kriteriju: U ovom slučaju, RCL lista sastoji se od p najboljih elemenata u smislu inkrementalnih troškova, gdje parametar p predstavlja maksimalan broj elemenata liste.
- Vrijednosno baziranom kriteriju: To je najčešće korištena strategija. Sastoji se od odabira rješenja koja su bolja od zadane vrijednosti praga.

U svakoj iteraciji, slučajni element je pokupljen s RCL liste. Nakon što je element ugrađen u djelomično rješenje, RCL lista je ažurirana.

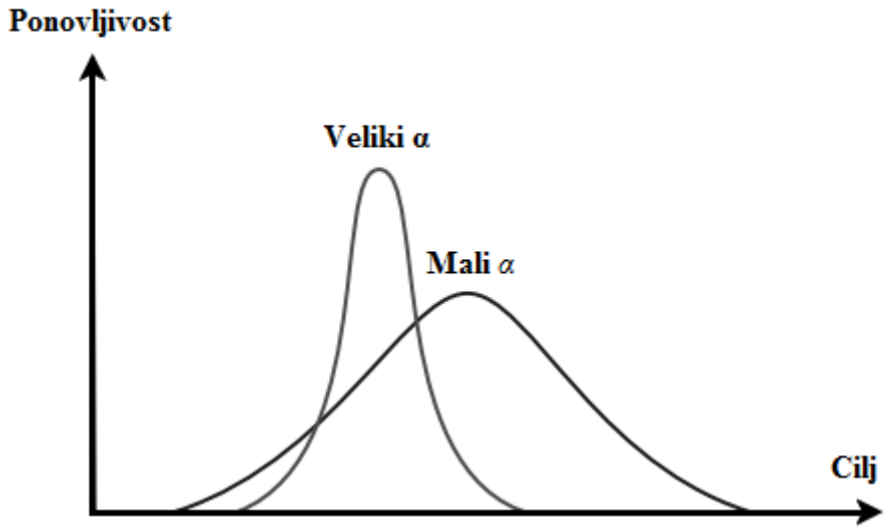
3.4.2. Korak lokalnog pretraživanja

S obzirom da rješenje dobiveno postupkom izgradnje nije nužno i lokalni optimum, korisno je provesti korak lokalnog pretraživanja, u kojem se poboljšava rješenje dobiveno u koraku izgradnje. Tradicionalno, primjenjuje se jednostavan algoritam lokalnog pretraživanja. Ipak, mogu se koristiti i druge metaheuristike bazirane na jednom rješenju, kao što su tabu pretraživanje, simulirano kaljenje i drugo.

3.4.3. Parametar α

Glavni parametar GRASP metode je parametar α ($\alpha \in [0, 1]$). On definira kompromis između eksploatacije (pojačavanja) i istraživanja (diversifikacije) prostora. U stvari, male vrijednosti α omogućuju veće vrijednosti prosječne kvalitete elemenata RCL liste i omogućuju poticanje eksploatacije. Kada je $\alpha = 1$, algoritam je deterministički pohlepni algoritam, a kada je $\alpha = 0$, algoritam je slučajni pohlepni algoritam. Mnogi eksperimentalni rezultati pokazuju

Gaussovu normalnu razdiobu za dobivena rješenja. Slika 3.2 prikazuje distribuciju rješenja kao funkciju α. Što je parametar α manji, veće su razlike između rješenja u fazi izgradnje i lošija je srednja kvaliteta rezultata.



Slika 3.2 Distribucija dobivenog lokalnog optimuma kao funkcija parametra α [3]

Budući da je parametar α kritičan parametar, mnoge strategije mogu se primijeniti za njegovo postavljanje [3]:

- Statička strategija: U ovoj strategiji, vrijednost α se postavlja na konstantnu vrijednost prije potrage. Najčešće se koristi raspon α koji varira između 0,7 i 0,9.
- Dinamička strategija: Ova strategija postavlja vrijednost parametra α nasumično za svaku iteraciju GRASP metaheuristike. Na primjer, može se koristiti uniformna razdioba [0,7; 0,9] ili padajuća neuniformna razdioba.
- Prilagodljiva strategija: U ovoj strategiji, vrijednost parametra α je samo-podesiva. Vrijednost α se automatski ažurira tijekom procesa pretraživanja memorije pretraživanja.

Dodatno uz parametre vezanih s nasumičnom pohlepnom heuristikom, GRASP metaheuristika će naslijediti parametre ugrađene metaheuristike zasnovane na jednom rješenju. Što je veća razlika između rješenja u fazi izgradnje, veća je razlika dobivenog lokalnog optimuma. Što je veća razlika između početnih rješenja, to je bolje najbolje pronađeno rješenje i duže je vrijeme računanja.

3.5. Monte Carlo metoda

Metodom Monte Carlo naziva se bilo koji način rješavanja problema koji se oslanja na generiranju velikog broja slučajnih brojeva te promatranje udjela tih brojeva koji pokazuju željena svojstva [15]. Metodu Monte Carlo je 1946. godine osmislio Stanislaw Ulam dok je radio na razvoju nuklearnog oružja u nacionalnom laboratoriju u Los Alamosu, a ime je dobila po kasinima Monte Carla. Vrijednost metode je ubrzo prepoznata od strane John von Neumanna koji je napisao program za prvo elektroničko računalo, ENIAC (eng. ENIAC - *Electronic Numerical Integrator And Computer*), gdje je probleme neutronske difuzije rješavao baš metodom Monte Carlo. Algoritam se može opisati na sljedeći način (Algoritam 3.5):

Algoritam 3.5 Algoritam Monte Carlo [15]

1. Matematički modelirati proces
2. Pronaći varijable čije vrijednosti nisu potpuno izvjesne
3. Odrediti funkcije gustoće koje dobro opisuju učestalosti kojima slučajne varijable poprimaju svoje vrijednosti
4. Ukoliko među varijablama postoje korelacije, napraviti matricu korelacija
5. U svakoj iteraciji svakoj varijabli dodijeliti slučajnu vrijednost proizašlu iz funkcije gustoće uzimajući u obzir matricu korelacije
6. Izračunati izlazne vrijednosti i spremiti rezultate
7. Korake 5 i 6 ponavljati *n* puta
8. Statistički analizirati rezultate simulacije

Vrijednost algoritma leži u tome što kao rezultat daje sve moguće ishode, ali i vjerojatnosti pojavljivanja svakog od tih ishoda. Nadalje, nad rezultatima Monte Carlo simulacije je moguće provesti analizu osjetljivost kako bi se identificirali čimbenici koji najviše utječu na ishod procesa kako bi se njihov utjecaj ograničio ili naglasio, ovisno o njihovoj prirodi.

4. METAHEURISTIKA - GLOBALNE METODE

Ovo poglavlje prikazuje kako prirodna selekcija može biti korištena za rješavanje problema optimizacije. Postupci opisani u poglavlju nazivaju se strategijom evolucije. Metode u poglavlju su podijeljene u dvije skupine: mikroskopske, koje se odnose na genetske algoritme, te makroskopske, koje simuliraju evoluciju određene vrste živog bića, kao što su mravi, pčele, krijesnice ili pojedinci dinamičke populacije s kognitivnom i društvenom svijesti.

U poglavlju su prikazane sljedeće metode: genetički algoritmi, optimizacija kolonijom mrava, optimizacija rojem čestica te optimizacija kolonijom pčela.

4.1. Genetski algoritam

Prve koncepte optimizacije, koji se odnose na genetski mehanizam, uveo je 1975. godine John Holland [16]. Također, on je uveo i pojam genetskog algoritma (eng. GA - *Genetic Algorithm*). Hollandov cilj nije bio dizajnirati algoritam koji će rješavati specifični problem, nego formalno proučavati fenomen adaptacije koji se zbiva u prirodi i razviti put po kojem mehanizam prirodne adaptacije može biti uključen u računalni sustav. Dakle, GA je heuristička metoda optimiranja koja imitira prirodni evolucijski proces. Evolucija je proces pretraživanja prostora rješenja. Živa bića se tijekom evolucije prilagođavaju uvjetima u prirodi, tj. svojoj životnoj okolini. Analogija evolucije kao prirodnog procesa i genetskog algoritma kao metode optimiranja, očituje se u procesu selekcije i genetskim operatorima. Mehanizam odabira nad nekom vrstom živih bića u evolucijskom procesu čine okolina i uvjeti u prirodi. U GA-u ključ selekcije je funkcija cilja, koja na određeni način predstavlja problem koji se rješava. Kao što su okolina i uvjeti u prirodi ključ selekcije nad nekom vrstom živih bića, tako je i funkcija cilja ključ selekcije nad nekom populacijom rješenja u GA-u. U prirodi, jedinka koja se najbolje prilagodi uvjetima i okolini u kojoj živi ima najveću vjerojatnost preživljavanja i parenja, a time i prenošenja svog genetskog koda na svoje potomke. Za GA jedno rješenje je jedna jedinka. Selekcijom se odabiru dobre jedinke koje se prenose u sljedeću populaciju, a manipulacijom gena stvaraju se nove jedinke. Takav ciklus selekcije, reprodukcije i manipulacije gena jedinki se ponavlja sve dok nije zadovoljen uvjet zaustavljanja. Konačan rezultat je populacija jedinki, odnosno populacija potencijalnih rješenja. Najbolja jedinka, dobivena u zadnjoj iteraciji, predstavlja rješenje optimiranja. U nastavku je dan prikaz algoritma (Algoritam 4.1).

Algoritam 4.1 Algoritam genetskog algoritma [17]

Generiraj početnu populaciju potencijalnih rješenja *p* ;
Sve dok nije zadovoljen uvjet zaustavljanja ;
    Selektiraj bolje jedinke ;
    Križaj bolje jedinke ;
    Mutiraj djecu boljih jedinki ;
Ispiši rješenje

4.1.1. Generiranje početne populacije

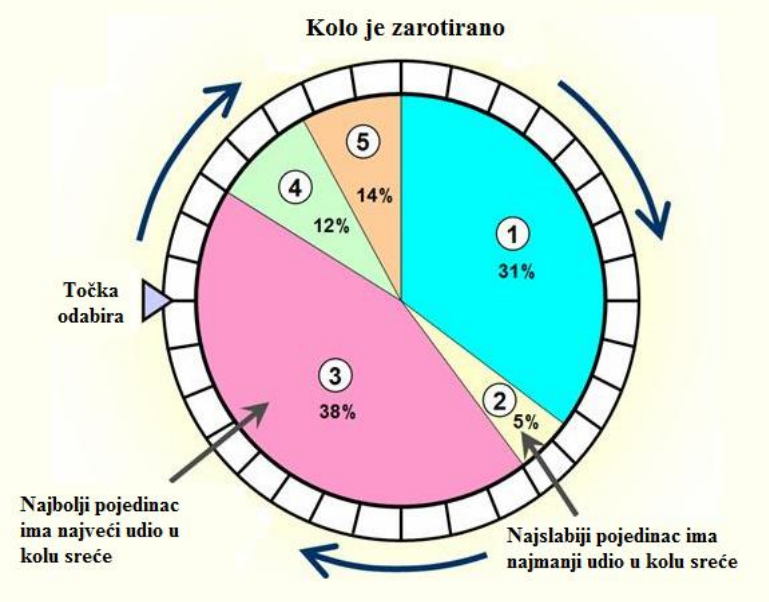
Uniformno generiranje početne populacije je postupak kojim se generira početna populacija u kojoj su sve jedinke iste [17]. U početku, populacija stvorena na ovaj način sporo konvergira rješenju. Neuniformno generiranje se može stvoriti na više načina, slučajnim generiranjem jedinki ili stvaranjem populacije od nekih već prije dobivenih jedinki, koje se onda algoritmom usavršavaju, ili nekom kombinacijom ovih načina.

4.1.2. Selekcija

Selekcija je proces odabira dva roditelja iz populacije jedinki za postupak križanja, tj. odabiru se jedinke koje će sudjelovati u reprodukciji te tako prenijeti svoj genetski materijal na sljedeću generaciju [17]. Cilj selekcije je istaknuti bolje pojedince u populaciji u nadi, da će njihovi potomci dati bolje rezultate. Kako i loše jedinke mogu sadržavati dobre i korisne gene, potrebno je i njima omogućiti kakvu takvu vjerojatnost za razmnožavanjem. Naravno, bolje jedinke trebaju imati veće vjerojatnosti za tim. Vrste selekcija koje su opisane u ovom radu su: kolo sreće, rang selekcija i k-turnirska selekcija.

4.1.2.1. Kolo sreće

Jedna od tradicionalnih selekcijskih tehnika je selekcija kolom sreće (eng. RWC - *Roulette Wheel Selection*) [18]. Svakom pojedincu je dodijeljen dio kola sreće, gdje je veličina tog djela proporcionalna njegovoj formi. Kolo sreće se okreće *N* puta, gdje je *N* broj pojedinaca (kromosoma) u populaciji. Pri svakom okretanju, pojedinac pokraj oznake „točka odabira“ (Slika 4.1) je selektiran za parenje u sljedećoj generaciji. Stoga, očekivana vrijednost svakog pojedinca je vrijednost funkcije cilja pojedinca podijeljena s ukupnom sumom funkcija cijele populacije.

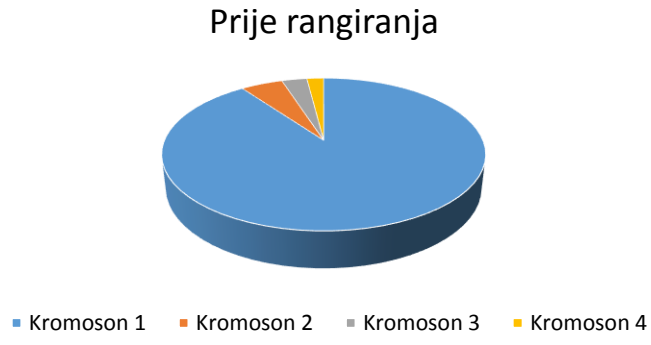


Slika 4.1 Selekcija kolom sreće [18]

Na prethodnoj slici je ilustrativno prikazan princip selekcije. Što je veća površina pojedinog pojedinca u kolu sreće, to je i veća vjerojatnost odabira baš tog pojedinca. Redom, pojedinci 1, 2, 3, 4 i 5 imaju vjerojatnost odabira 31 %, 5 %, 38 %, 12 % i 14 %. Odnosno najveću vjerojatnost odabira ima pojedinac 3 s 38 %.

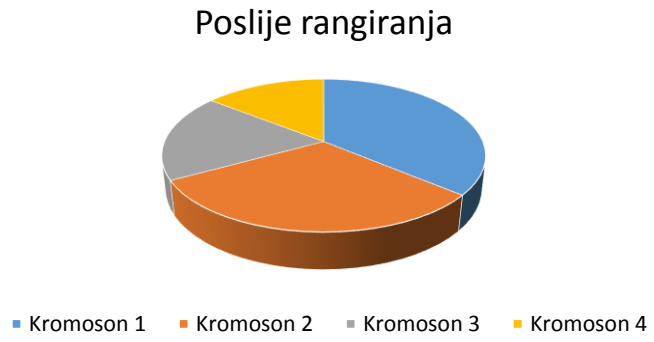
4.1.2.2. Rang selekcija

Prethodna selekcija ima problema kada se vjerojatnosti pojedinaca veoma razlikuju. Na primjer, ako najbolji pojedinac (kromosom) ima 90 % ukupne površine cijelog kola onda će ostali pojedinci imati vrlo malo izgleda da budu selektirani (Slika 4.2).



Slika 4.2 Situacija prije rangiranja [19]

Rang selekcija je alternativna metoda čiji je cilj spriječiti prebrzu konvergenciju. Pojedinci u populaciji su rangirani prema funkciji, a očekivane vrijednosti svakog pojedinca ovise više o rangu, nego o njihovoj vrijednosti funkcije. Najgoru funkciju ima rang 1, a najbolju rang *N*.



Slika 4.3 Situacija poslije rangiranja [19]

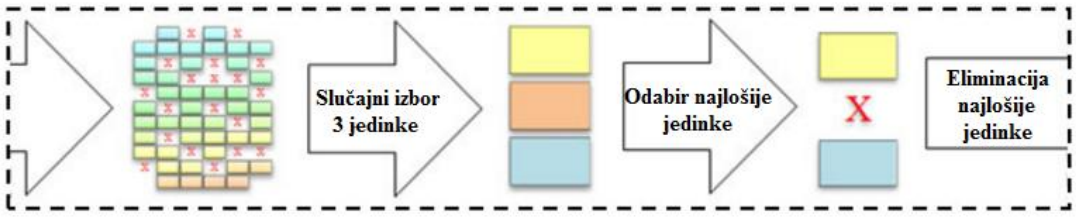
Nakon toga, svi kromosomi imaju izgleda da budu selektirani (Slika 4.3). No, ova metoda može dovesti do sporije konvergencije, jer se najbolji kromosom ne razlikuje toliko od onih drugih.

4.1.2.3. K-turnirska selekcija

K-turnirska selekcija je rangirajuća vrsta selekcije koja u svakom koraku za generiranje nove populacije odabire *k* jedinki iz stare populacije, rangira ih, te najbolju među njima stavlja u bazen za reprodukciju nad kojom će se onda izvršiti genetski operatori reprodukcije [17]. Čest je slučaj odabira 3-turnirske selekcije, gdje se nasumično odabiru tri jedinke, i potom uzima najbolja od njih (Slika 4.4). Da bi se dobila dva roditelja koja će se potom križati, potrebno je obaviti dva turnira.

Slika 4.4 Primjer jednog od koraka generacijske 3-turnirske selekcije [17]

Za razliku od generacijske, eliminacijska turnirska selekcija odabire *k* jedinki te briše najslabiju od njih iz trenutne populacije, a može je nadomjestiti djetetom preostale dvije jedinke. Na slici 4.5 prikazana je 3-turnirska eliminacijska selekcija.



Slika 4.5 Primjer jednog od koraka eliminacijske 3-turnirske selekcije [17]

U praksi se često koristi i dodatno pojednostavljenje 3-turnirske selekcije. Da bi se dobila dva roditelja za križanje, nasumice se iz populacije izvuku 3 jedinke i uzmu se dvije bolje, a najlošija jedinka se odbaci i zamjeni nastalim djetetom. Time se umjesto obavljanja tri koraka turnira sve riješilo samo jednim.

4.1.3. Križanje

Križanje je genetski operator koji imitira prirodni proces križanja [17]. U njemu sudjeluju dvije jedinke iz populacije, koje su odabrane nekim od postupaka selekcije, te se križanjem njihovog gena dobiva jedna ili dvije nove jedinke. Pošto je novonastala jedinka dobivena križanjem dviju dobrih jedinki, i njena svojstva bi trebala biti takva, ako ne i bolja. Križanje se može izvesti na nekoliko načina. Križanjem se brzo postiže konvergencija optimumu, ali postoji opasnost da on bude lokalni. Najjednostavniji način križanja je križanje s jednom točkom prekida kada se kromosom lomi u jedno točki te spaja s dijelom drugog slomljenog na istom mjestu (Tablica 4.1).

Tablica 4.1 Primjer križanja jedinki s jednom točkom prekida

Roditelj A	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
Roditelj B	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**
Dijete 1	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**
Dijete 2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**

Križanje s više točaka prekida obavlja se na sličan način. Jedina je razlika u tome što se kromosomi lome u više točaka, a zatim se dijelovi A i B naizmjenice spajaju. Takav slučaj prikazuje tablica 4.2.

Tablica 4.2 Primjer križanja jedinki s više točaka prekida

Roditelj A	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
Roditelj B	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**
Dijete 1	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**
Dijete 2	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**

Krajnji slučaj križanja s više točaka je uniformno križanje, tj. križanje s b - 1 prekidnih točaka na kromosomu s b bitova. Kod uniformnog križanja, dijelovi kromosoma se ne spajaju naizmjenice već se za svaki gen računa da li je naslijeđen od prve ili druge jedinke koja sudjeluje u križanju, gdje je vjerojatnost nasljeđivanja gena od svakog roditelja $p = 0{,}5$.

4.1.4. Mutacija

Mutacija je genetski algoritam, koji sprječava da algoritam bude zaglavljen u lokalnom minimumu. Mutacija djeluje na jedan kromosom pri čemu ga izmjenjuje. Mutacija može unijeti novo svojstvo u vrstu ili obnoviti izgubljeno svojstvo vrste. To je svojevrsno osiguranje od protiv nepovratnog gubitka genetskog materijala. Ukoliko je vjerojatnost mutacije jednaka jedan, tada je algoritam zapravo algoritam traženja slučajnom pretragom prostora rješenja. Što je vjerojatnost mutacije manja, to je manja vjerojatnost pronalaska globalnog optimuma. Ukoliko je vjerojatnost mutacije približno nuli, algoritam će vjerojatno završiti u lokalnom optimumu. U tablici 4.3 dan je primjer jednostavne mutacije nad slučajno odabranom genu, gdje crveno označen gen predstavlja mutaciju.

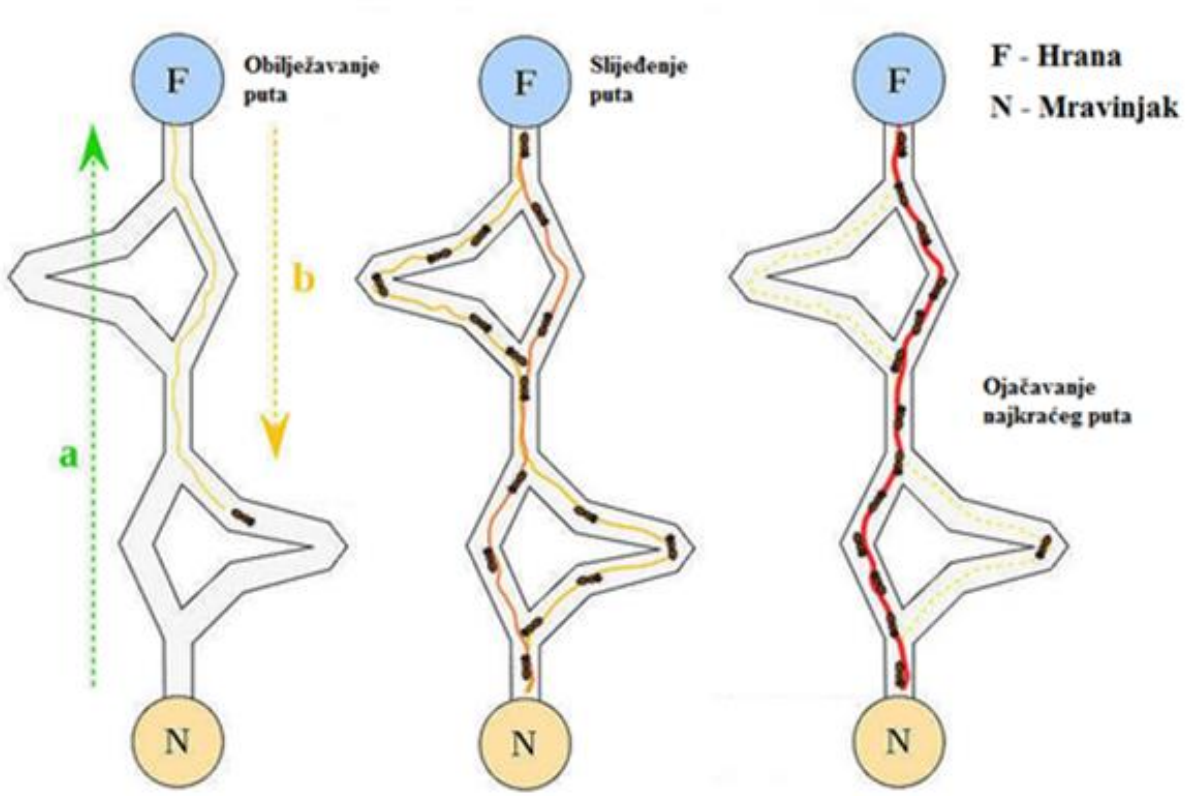
Tablica 4.3 Primjer jednostavne mutacije nad slučajno odabranom genu

Kromosom prije mutacije	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
Kromosom poslije mutacije	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**

Mutacija se može vršiti nad jednim slučajno odabranim genom, nad genima odabranima pomoću slučajno generirane maske ili nad svim genima - potpuna mutacija [17]. Jednostavna mutacija je najjednostavniji oblik mutacije, gdje se odabranom genu izmjenjuje vrijednost. Miješajuća mutacija izmiješa pozicije dva ili više odabranih gena, potpuno miješajuća mutacija slučajno generira vrijednosti odabranih gena, dok invertirajuća miješajuća mutacija invertira vrijednost gena.

4.2. Optimizacija kolonijom mrava

Optimizacija kolonijom mrava (eng. ACO - *Ant Colony Optimization*) je predložena od strane Marca Doriga [20]. Osnova ideja ACO-a je imitiranje kooperativnog ponašanja mrava za rješavanje problema optimizacije. ACO algoritam je primijenjen na probleme kombinatornog optimiranja, gdje je postigao uspjehe u rješavanju različitih problema (npr. raspoređivanja, usmjeravanja i prijenosa). Glavni interes za imitiranjem mrava je u tome što oni svojim ponašanjem i kolektivnim radom obavljaju kompleksne zadatke, kao što su transport hrane i pronalaženje najkraćeg puta do izvora hrane. ACO algoritam oponaša princip korištenja vrlo jednostavnog mehanizma komunikacije mrava, gdje je kolonija mrava u stanju pronaći najkraći put do hrane. Slika 4.6 prikazuje eksperiment s kolonijom mrava.



Slika 4.6 Eksperiment ACO-a [21]

Kolonija ima pristup izvoru hrane koja je povezana gnijezdom kolonije. Tijekom svojeg putovanja, mravi ostavljaju kemijsku tvar zvanu feromon. Feromon je olfaktivna (mirisna) i hlapljiva tvar koja izaziva društveni odgovor kod članova iste vrste. Uloga te tvari je navoditi druge mrave prema ciljanoj točki, u ovom slučaju prema hrani. Što je veća količina feromona na određenom putu, veća je i vjerojatnost da će mravi odabrati taj put. Količina ostavljenog feromona jednog mrava ovisi o količini hrane i duljini puta. Pošto je feromon hlapljiva tvar ona ima opadajuće djelovanje tijekom vremena. Kako je prikazano na slici 4.6, kada se mrav suočava s prvim račvanjem (odnosno drugim), postoji jednaka vjerojatnost da će izabrati put desno, odnosno put lijevo. Pošto je kod prvog račvanja lijeva staza kraća od desne (kod drugog račvanja desna staza je kraća od lijeve), to rezultira manjim vremenom putovanja mrava i višom razinom ostavljenog feromona. Što više mrava izabere kraći put, veća količina feromona je ostavljena te dolazi do pojave najkraćeg puta. Ova činjenica će se povećati isparavanjem feromona.

4.2.1. Algoritam ACO

U nastavku je dan prikaz algoritma optimizacije kolonijom mrava (Algoritam 4.2).

Algoritam 4.2 Algoritam optimizacije kolonijom mrava [3]

Inicijalizacija feromonskog traga ;
Ponavljaj
    Za svakog mrava **Učini**
    Izgradnja rješenja pomoću feromonskog traga ;
    Ažuriraj feromonski trag ;
        Isparavanje ;
        Ojačavanje ;
Dok nije zadovoljen kriterij zaustavljanja
Ispiši najbolje rješenje ili populaciju rješenja

Algoritam se uglavnom sastoji od dva iterativna koraka, koraka izgradnje rješenja i koraka ažuriranja feromona [3]:

- Izgradnja rješenja: Na umjetne mrave se može gledati kao na stohastičke pohlepne postupke koji grade rješenje dodavanjem komponenti rješenja sve dok se ne postigne cjelovito rješenje. Ovaj iterativni postupak uzima u obzir:
    - Feromonski trag: Feromonski trag pamti karakteristike dobrih generiranih rješenja, koja će voditi izgradnju novih rješenja. Feromonski trag se dinamički mijenja tijekom pretraživanja, kako bi održao stečena znanja. Može se reći da feromonski trag predstavlja memoriju cijelog procesa pretraživanja.
- Ažuriranje feromona: Ažuriranje se provodi uz pomoć generiranih rješenja. Pravilo ažuriranja se provodi kroz dvije faze:
    - Faza isparavanja: U ovoj fazi dolazi do automatskog smanjenja razine feromona. Cilj isparavanja je izbjeći preranu konvergenciju prema dobrim rješenjima i zatim poticati diversifikaciju u prostoru pretraživanja (istraživanje).
    - Faza ojačavanja: U ovoj fazi se feromonski trag ažurira u skladu s generiranim rješenjem. Koriste se tri različite strategije: *online* korak po korak, *online* odgođeno i *offline* ažuriranje feromona.

Prethodni koraci se izvršavaju sve dok se ne postigne kriterij zaustavljanja. Kriterij zaustavljanja može biti: broj iteracija koji se mora izvesti, postavljena vremenska granica rada, zadani broj rješenja za procjenu itd.

4.3. Optimizacija rojem čestica

Optimizacija rojem čestica (eng. PSO - *Particle Swarm Optimization*) je još jedna stohastička metoda bazirana na populaciji rješenja nadahnuta inteligencijom roja. Ova metoda razvijena je od strane Jamesa Kennedya i Russell Eberharta 1995. godine [22]. Metoda oponaša društveno ponašanje prirodnih organizama, kao što su ribe i ptice u potrazi za hranom.

PSO metoda dijeli mnoge sličnosti s evolucijskim tehnikama, kao što su genetski algoritmi. Sustav rješenja je kreiran populacijom od slučajnih rješenja i zatim se potraga za optimumom vrši ažuriranjem rješenja. Međutim, za razliku od GA, PSO nema evolucijske operatore kao što su križanje i mutacija. U PSO optimizaciji, potencijalna rješenja, koja se nazivaju čestice, lete kroz prostor problema prateći pritom trenutne najbolje čestice. Svaka čestica vodi evidenciju svojih koordinata u prostoru problema. Te koordinate su povezane s najboljim rješenjem što je do sada postignuto od strane te čestice. Također, vrijednost tog najboljeg rješenja je pohranjena, koja se naziva *pbest* [23]. Još jednu dobru vrijednost, koju prati roj čestica, je najbolja vrijednost koja je postignuta od strane cijelog roja čestica. Ta lokacija zove se *lbest*. Kada čestica zauzme cijelu populaciju, najbolja vrijednost je globalno najbolja i ta vrijednost se označuje kao *gbest*. Koncept PSO sastoji se od mijenjanja brzine svake čestice prema njenoj *pbest* i *lbest* lokaciji u svakom koraku vremena, odnosno u svakoj iteraciji.

U osnovnom modelu, roj se sastoji od N čestica koje lete u D-dimenzionalnom prostoru odlučivanja. Svaka čestica i je mogući kandidat rješenja problema, a predstavljena je vektorom x_i u prostoru rješenja. Svaka čestica ima svoj položaj i brzinu preko kojih se određuje smjer i korak čestice. Prilikom optimizacije optimizacija iskorištava suradnju između čestica. Uspjeh pojedinih čestica utjecat će na ponašanje drugih čestica. Svaka čestica uspješno prilagođava svoj položaj x_i prema globalnom optimumu, imajući u vidu sljedeća dva faktora: najbolji ostvaren položaj koju je ostvarila promatrana čestica (*pbest*) označen s $p_i = (p_{i1}, p_{i2}, \ldots p_{iD})$ i najboljeg položaja ostvarenog od strane cijelog roja (*gbest*) označen s $p_g = (p_{g1}, p_{g2}, \ldots p_{gD})$. Vektor $p_g - x_i$ predstavlja razliku između trenutnog položaja čestice i i najboljeg položaja njenog susjedstva.

Za svaku česticu mora biti definirano susjedstvo. To susjedstvo prikazuje društveni utjecaj među česticama. Postoje razne mogućnosti za definiranje takvog susjedstva. Tradicionalno, koriste se dvije metode:

- Metoda globalnog optimuma: U metodi globalnog optimuma, susjedstvo se definira kao cjelokupna populacija rješenja.

- Metoda lokalnog optimuma: U metodi lokalnog optimuma, susjedstvo čestica čine skup direktno povezanih čestica.

Roj čestica se može promatrati kao stanični automat gdje se ažuriraju stanice (čestice kod PSO). Svaka nova vrijednost stanice ovisi samo o staroj vrijednosti stanice i njenom okruženju, a sve stanice se ažuriraju pomoću istih pravila. Pri svakoj iteraciji, na česticu se primjenjuju sljedeće operacije: ažuriranje brzine, ažuriranje položaja i ažuriranje rješenja.

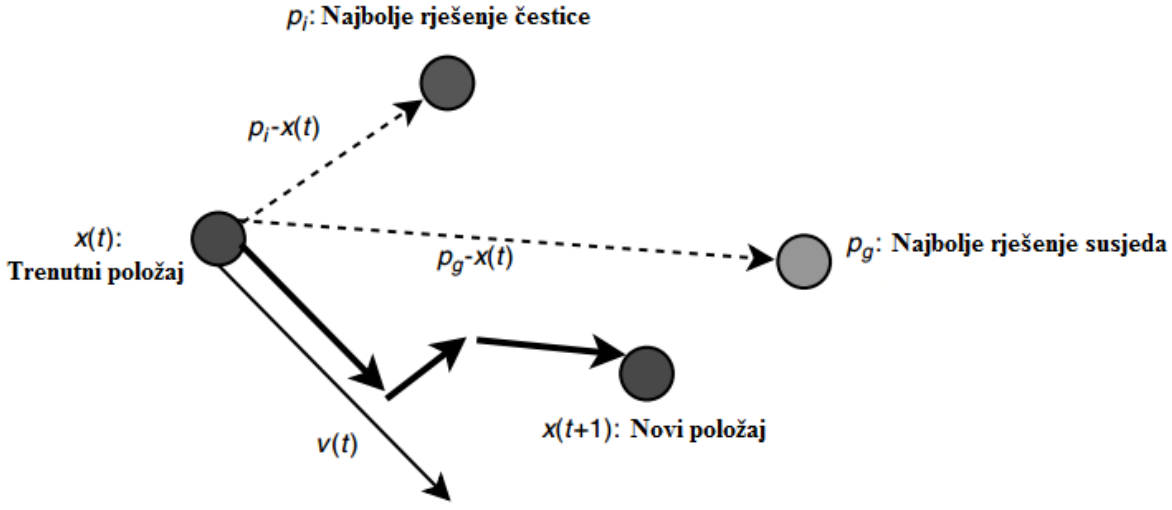
4.3.1. Ažuriranje brzine, položaja i rješenja

Brzina, koja određuje količinu promjene koja će se primijeniti na česticu, definirana je kao [3]:

$$v_{\mathrm{i}}(t) = v_{\mathrm{i}}(t-1) + \rho_1 C_1 \cdot [p_{\mathrm{i}} - x_{\mathrm{i}}(t-1)] + \rho_2 C_2 \cdot \left[p_{\mathrm{g}} - x_{\mathrm{i}}(t-1)\right] \tag{8}$$

gdje su ρ_1 i ρ_2 dvije slučajne varijable u rasponu [0, 1]. Konstante C_1 i C_2 predstavljaju faktor učenja. Oni predstavljaju privlačenje koje čestica ima prema vlastitom uspjehu ili prema uspjehu svojih susjeda. Parametar C_1 je kognitivni faktor učenja koji predstavlja privlačenje koje čestica ima prema vlastitom uspjehu, dok je parametar C_2 faktor socijalnog učenja koji predstavlja privlačenje koje čestica ima prema uspjehu svojih susjeda. Brzinom se prikazuje smjer i udaljenost koju čestica treba proći (Slika 4.7). Ova formula je odraz temeljnog aspekta ljudske društvenosti u kojoj društveno-psihološka sklonost pojedinaca utječe na uspjeh drugih pojedinaca. Točka oko koje će čestica kružiti definirana je kao prosjek od p_{i} i p_{g}:

$$\frac{\rho_1 \cdot p_{\mathrm{i}} + \rho_2 \cdot p_{\mathrm{g}}}{\rho_1 + \rho_2} \tag{9}$$



Slika 4.7 Kretanja čestice i ažuriranje brzine [3]

Tijekom ažuriranja brzine, koeficijent inercije *w* se obično dodaje na prethodnu brzinu:

$$v_{\mathrm{i}}(t) = w \cdot v_{\mathrm{i}}(t-1) + \rho_1 \cdot [p_{\mathrm{i}} - x_{\mathrm{i}}(t-1)] + \rho_2 \cdot \left[p_{\mathrm{g}} - x_{\mathrm{i}}(t-1)\right] \tag{10}$$

Koeficijent inercije *w* kontrolira utjecaj prethodne brzine na trenutnu. Pri većim vrijednostima koeficijenta inercije, utjecaj prethodnih brzina će biti mnogo veći. Dakle, koeficijent inercije predstavlja kompromis između globalnog istraživanja i lokalne eksploatacije. Veći koeficijent inercije potiče globalno istraživanje, dok manji koeficijent inercije potiče lokalnu eksploataciju. Svaka čestica ažurira svoje koordinate u prostoru rješenja prema formuli (11) i tada prelazi na novi položaj:

$$x_{\mathrm{i}}(t) = x_{\mathrm{i}}(t-1) + v_{\mathrm{i}}(t) \tag{11}$$

Svaka čestica ažurira najbolje lokalno rješenje prema:

- ako je $f(x_{\mathrm{i}}) < pbest_{\mathrm{i}}$, onda je $p_{\mathrm{i}} = x_{\mathrm{i}}$

dok se globalno rješenje roja ažurira prema:

- ako je $f(x_{\mathrm{i}}) < gbest_{\mathrm{i}}$, onda je $g_{\mathrm{i}} = x_{\mathrm{i}}$

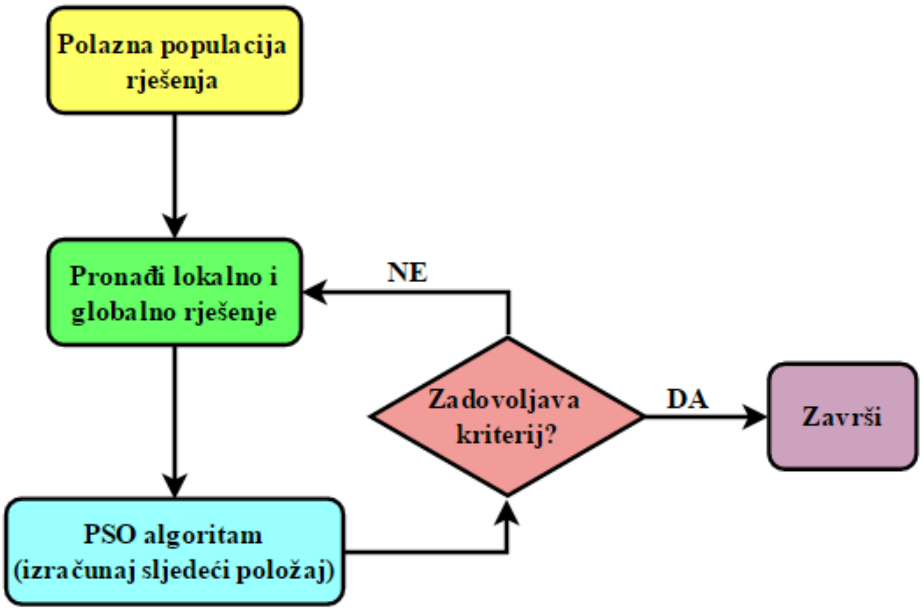
4.3.2. Algoritam PSO

U nastavku je dan prikaz algoritma optimizacije rojem čestica (Algoritam 4.3).

Algoritam 4.3 Algoritam optimizacije rojem čestica [3]

Početna inicijalizacija cijelog roja
Ponavljaj
Evaluacija $f(x_{\mathrm{i}})$;
Za sve čestice *i*
Ažuriraj brzinu**:**
$v_{\mathrm{i}}(t) = v_{\mathrm{i}}(t-1) + \rho_1 \cdot [p_{\mathrm{i}} - x_{\mathrm{i}}(t-1)] + \rho_2 \cdot \left[p_{\mathrm{g}} - x_{\mathrm{i}}(t-1)\right]$;
Pomak na novi položaj: $x_{\mathrm{i}}(t) = x_{\mathrm{i}}(t-1) + v_{\mathrm{i}}(t)$;
Ako je $f(x_{\mathrm{i}}) < f(pbest_{\mathrm{i}})$ **Onda** je $pbest_{\mathrm{i}} = x_{\mathrm{i}}$;
Ako je $f(x_{\mathrm{i}}) < f(gbest)$ **Onda** je $gbest = x_{\mathrm{i}}$;
Ažuriraj (x_{i}, v_{i}) ;
Do kriterija zaustavljanja

Koraci prikazanog algoritma su objašnjeni u prethodnom potpoglavlju *4.3.1. Ažuriranje brzine, položaja i rješenja*. Ovaj algoritam se može jednostavnije prikazati kao dijagram toka, kako je prikazano na slici 4.8.



Slika 4.8 Dijagram toka PSO algoritma [7, 25]

Dijagram toka sastoji se od polazne populacije rješenja, pronalaženja lokalnog i globalnog rješenja, izračunavanja sljedećeg položaja te zadovoljavanja kriterija zaustavljanja. Cijeli princip rada objašnjen je u ranijim potpoglavljima.

Kriterij zaustavljanja može biti: postignuti maksimalan broj iteracija ili pronađeno rješenje s zadovoljavajućom funkcijom cilja.

4.4. Optimizacija rojem pčela

Optimizacija rojem pčela je metaheuristička metoda temeljena na slučajnom pretraživanju populacije. Tehnika je izrađena na temelju pronađene analogije između prirodnog ponašanja pčela u potrazi za hranom i ponašanja optimizacijskih algoritama u potrazi za optimumom. Osnovna ideja metode je izgraditi sustav agenata (kolonije umjetnih pčela), koji će tražiti dobra rješenja različitih kombinatornih problema, istražujući načela koja koriste pčele tijekom skupljanja nektara. Umjetna kolonija pčela se obično sastoji od malog broja pojedinaca, ali svejedno principi metode su temeljeni na prirodnim sustavima. Metoda se temelji na umjetnim pčelama koje su u potrazi za provedivim rješenjima u prostoru pretraživanja. Kako bi se povećala kvaliteta proizvedenih rješenja, autonomne umjetne pčele surađuju i izmjenjuju informacije. Dijeljenjem dostupnih informacija i korištenjem kolektivnog znanja, umjetne pčele se koncentriraju na više obećavajuća područja te polako napuštaju manje obećavajuća rješenja. Korak po korak, umjetne pčele generiraju i/ili poboljšavaju svoja pronađena rješenja sve dok nije postignut unaprijed zadani kriterij zaustavljanja.

4.4.1. BCO algoritam

Kako se algoritam optimizacije kolonijom pčela (eng. BCO - *Bee Colony Optimization*) razvijao, autori su razvili dva različita pristupa [24]. Prvi pristup temelji se na koracima izgradnje gdje pčele grade rješenje korak po korak pa se stoga naziva konstruktivni BCO. Drugi pristup temelji se na poboljšanju cjelovitih rješenja kako bi se dobila što bolja konačna rješenja te se taj pristup naziva BCO_i. U nastavku su oba pristupa objašnjena kroz opći opis algoritma (Algoritam 4.4).

Algoritam 4.4 BCO algoritam [24]

Inicijalizacija: Učitaj parametre B i NC te kriterij zaustavljanja.
Učini sljedeće
(1) **Dodaj** (prazno) rješenje svakoj pčeli ;
(2) **Za** ($i = 0$; $i < NC$; $i + +$)
// prolaz naprijed
(a) **Za** ($b = 0$; $b < B$; $b + +$)
Za ($s = 0$; $s < f(NC)$; $s + +$) // broji poteze
(i) Evaluacija mogućih poteza ;
(ii) Odaberi jedan potez koristeći kolo sreće ;
// prolaz natrag
(b) **Za** ($b = 0$; $b < B$; $b + +$)
Evaluacija (djelomičnih/cjelovitih) rješenja pčela b ;
(c) **Za** ($b = 0$; $b < B$; $b + +$)
Odluka lojalnosti za pčelu b ;
(d) **Za** ($b = 0$; $b < B$; $b + +$)
Ako je b neopredijeljena, odaberi regrutera koristeći kolo sreće ;
(3) **Evaluacija** svih rješenja i pronalazak najboljeg. Ažuriraj x_{best} i $f(x_{best})$ dok se ne zadovolji kriterij zaustavljanja.
Ispiši x_{best} i $f(x_{best})$

Populacija agenata sastoji se od B pojedinaca (umjetnih pčela) koji su angažirani. Svaka umjetna pčela preuzima odgovornost za proizvodnju jednog rješenja. Algoritam se sastoji od dvije naizmjenične faze: prolaza naprijed i prolaza natrag [24]. Te dvije naizmjenične faze čine jedan korak BCO algoritma. Tijekom svake faze prolaza naprijed, sve pčele pretražuju prostor pretraživanja. Prostor pretraživanja se pretražuje prema unaprijed definiranom broju poteza, koji izgrađuju i/ili poboljšavaju rješenje čime se dobiva novo rješenje. Nakon što je dobiveno novo djelomično ili cjelovito rješenje, započinje se s drugom fazom prolaza natrag. U prolazu natrag, sve umjetne pčele dijele informacije o kvaliteti njihovih rješenja, bilo da se radi o djelomičnim ili cjelovitim rješenjima. U prirodi, pčele bi se vratile u košnicu, izvele plesni ritual

koji bi obavijestio druge pčele o količini hrane koju su otkrile i na kolikoj udaljenosti od košnice se nalazi. U algoritmu pretraživanja, pronalazak kvalitetnog rješenja se obavlja izračunavanjem funkcije cilja za svako djelomično i/ili cjelovito rješenje. Nakon što je izvršena evaluacija svih rješenja, svaka pčela odlučuje s određenom vjerojatnošću da li će ostati vjerna svojem rješenju ili ne. Pčele s boljim rješenjem imaju veće šanse da zadrže i reklamiraju svoja rješenja. Za razliku od pčela u prirodi, umjetne pčele koje su vjerne svojem rješenju su u isto vrijeme i regruteri, tj. njihova rješenja će razmatrati i druge pčele kao potencijalno dobra. Jednom kad umjetna pčela odbaci svoje rješenje, ona postaje neopredijeljena i mora odabrati jedno od reklamirajućih rješenja. I ova odluka se donosi s određenom vjerojatnošću, tako da bolja reklamirajuća rješenja imaju veće mogućnosti da budu izabrana za daljnja istraživanja. Na taj način, u svakom prolazu unatrag, sve pčele su podijeljene u dvije skupine. Prvu skupine čine pčele regruteri *RE*, a drugu neopredijeljene pčele *B - RE*. Vrijednosti *RE* i *B - RE* se izmjenjuju od jednog prolaza natrag do drugog prolaza natrag. U sljedećem prolazu naprijed, prema konstruktivnom BCO-u, svaka pčela dodaje novu različitu komponentu na ranije generirano djelomično rješenje, dok kod BCO_i-a pčele izmjenjuju neke dijelove cjelovitog rješenja kako bi se postigla rješenja što veće kvalitete.

Ove dvije faze, naprijed i nazad, se izmjenjuju naizmjenično *NC* puta, odnosno, dok svaka pčela ne izgradi svoje rješenje ili ne obavi izmjenu rješenja. *NC* je parametar koji se koristi za definiranje učestalosti razmjene informacija između pčela. Kada se izvrše *NC* koraci, najbolje rješenje od svih mogućih *B* rješenja se određuje. To rješenje se zatim koristi za ažuriranje globalno najboljeg rješenja i iteracija BCO algoritma je završena. U ovom trenutku, sva *B* rješenja se brišu i nova iteracija može biti započeta. BCO algoritam vrši iteraciju za iteracijom sve dok kriterij zaustavljanja nije ispunjen. Mogući kriteriji zaustavljanja mogu biti, na primjer, maksimalan broj iteracija ili maksimalan broj iteracija bez poboljšanja vrijednosti funkcije cilja. Na kraju, najbolje pronađeno rješenje se daje kao konačno. To rješenje je ujedno i globalno rješenje.

4.4.1.1. Odluka lojalnosti pčele

Nakon što je završena faza naprijed, svaka pčela mora odlučiti hoće li ostati vjerna već otkrivenom rješenju ili ne. Ova odluka zavisi o kvaliteti vlastitog rješenja u odnosu na kvalitetu rješenja drugih pčela. Vjerojatnost da je pčela *b* lojalna svome prethodno pronađenom djelomičnom/cjelovitom rješenju na početku nove faze unaprijed je [24]:

$$p_{\mathrm{b}}^{\mathrm{u}+1} = \mathrm{e}^{-\frac{Q_{\max} - Q_{\mathrm{b}}}{u}}, \quad b = 1, 2, \ \ldots \ B \tag{12}$$

Gdje je:

- Q_b - normalizirana vrijednost funkcije cilja (parcijalnog/cjelovitog rješenja) pčele *b*
- Q_{max} - maksimalna vrijednost od svih normaliziranih vrijednosti
- *u* - brojač fazi unaprijed (*u* = 1, 2, ... *NC*)

Normalizacija se provodi na dva načina, ovisno da li se traži minimum ili maksimum funkcije cilja. Ako C_b označava vrijednost funkcije cilja *b*-tog pčelinjeg rješenja, onda se Q_b za slučaj minimiziranja izračunava kao:

$$Q_b = \frac{C_{max} - C_b}{C_{max} - C_{min}}, \quad b = 1, 2, \ \dots \ B \tag{13}$$

gdje su C_{min} i C_{max} vrijednosti djelomičnih/cjelovitih rješenja vezanih uz minimalnu i maksimalnu vrijednost funkcije cilja, dobivenih od strane svih pčela. U slučaju maksimuma formula glasi:

$$Q_b = \frac{C_b - C_{max}}{C_{max} - C_{min}}, \quad b = 1, 2, \ \dots \ B \tag{14}$$

Upotrebom jednadžbe (12) i generatora slučajnih brojeva, svaka umjetna pčela odlučuje da li će postati neopredijeljeni sljedbenik ili nastaviti istraživati vlastito rješenje. Ako je izabrani slučajni broj manji od izračunate vrijednosti, onda pčela ostaje vjerna svom rješenju. Inače, ako je izabrani slučajni broj veći od vjerojatnosti p_b^{u+1}, pčela postaje neopredijeljena.

4.4.1.2. Proces regrutacije

Nakon što je pčela postala neopredijeljena, pčela mora odlučiti kojeg regrutera će dalje slijediti, uzimajući u obzir kvalitetu reklamirajućih rješenja. Vjerojatnost odabira djelomičnog/cjelovitog rješenja pčele *b* od strane drugih pčela je:

$$p_b = \frac{Q_b}{\sum_{k=1}^{RE} Q_k}, \quad b = 1, 2, \ \dots \ RE \tag{15}$$

Gdje Q_k predstavlja normaliziranu vrijednost funkcije cilja *k*-tog reklamiranog rješenja, a *RE* označava broj regrutera. Koristeći jednadžbu (15) i generator slučajnih brojeva, svaka neopredijeljena pčela se pridružuje jednom regruteru preko kola sreće (Vidi poglavlje *4.1.2.1. Kolo sreće*).

4.4.1.3. Primjena BCO-a

BCO je primijenjen na sljedeće klase problema [24]:

- Usmjeravanje:
    - problem putujućeg trgovca

  - problem usmjeravanja vozila
  - usmjeravanje i dodjeljivanje zadataka u svim optičkim mrežama.
- Lokacija:
  - *p*-medijan problem,
  - problem lokacije senzora na autocesti,
  - problem lokacije inspekcijskih postaja na mreži prometa,
  - problem prekrivanja adrese i
  - problem lokacije *p*-centra ili problem distribucije resursa.
- Raspored:
  - izrada rasporeda samostalnih zadataka homogenim višeprocesorskim sustavima,
  - izrada rasporeda vožnje,
  - izrada rasporeda rada,
  - raspoređivanje zadataka u računalnim mrežama te mnogi drugi.
- Medicina u vezi s kemijom:
  - terapije raka i
  - kemijski proces optimizacije.
- Mreže:
  - dizajn mreža.
- Kontinuirani i mješoviti problem optimiranja:
  - minimiziranje numeričke funkcije te
  - problem zadovoljenja u probabilističkoj logici.

4.4.2. ABC algoritam

Algoritam kolonije umjetnih pčela (eng. ABC - *Artificial Bee Colony*) je metaheuristički algoritam baziran na populaciji rješenja kojeg je predstavio Tereshko 2005. godine [25]. Algoritam je inspiriran inteligentnim ponašanjem pčela za vrijeme potrage za hranom koje izvršavaju razne zadatke kroz društvenu suradnju.

U ABC algoritmu postoje tri vrste pčela: zaposlene pčele, pčele promatrači i pčele izviđači. Zaposlene pčele iskorištavaju izvor hrane, nose podatke o izvoru hrane natrag u košnicu te dijele te informacije s pčelama promatračima. Pčele promatrači čekaju u košnici kako bi s njima zaposlene pčele podijelile informacije o otkrivenim izvorima hrane, dok su pčele izviđači u stalnoj potrazi za novim izvorima hrane u blizini košnice. Zaposlene pčele dijele informacije o

izvorima hrane plešući u određenom plesnom prostoru unutar košnice. Priroda plesa je proporcionalna sadržaju nektra izvora hrane koju iskorištava plešuća (zaposlena) pčela. Pčele promatrači promatraju brojne plesove zaposlenih pčela te odabiru izvor hrane s određenom vjerojatnošću da je kvaliteta tog izvora hrane proporcionalna plesu zaposlene pčele. Dakle, dobri izvori hrane će privući više pčela promatrača u odnosu na lošije izvore. Nakon što pčela promatrač odabere jedan od mogućih izvora hrane, ona postaje zaposlena pčela. Kad god je izvor hrane u potpunosti iskorišten sve zaposlene pčele povezane s njim napuštaju taj izvor te postaju pčele izviđači. Stoga, pčele izviđače možemo vizualizirati kao radnike koji obavljaju poslove istraživanja, dok zaposlene pčele i pčele promatrače možemo vizualizirati kao radnike koji obavljaju poslove eksploatacije (iskorištavaju izvor hrane). Glavni koraci algoritma prikazani su u nastavku (Algoritam 4.5).

Algoritam 4.5 ABC algoritam [25]

Inicijalizacija populacije
Ponavljaj
Postavi zaposlene pčele na njihove izvore hrane
Postavi pčele promatrače na izvore hrane, ovisno o količini nektra
Pošalji pčele promatrače na područje pretraživanja za otkrivanje novih izvora hrane
Zapamti najbolji izvor hrane do sada pronađen
Sve dok se ne zadovolji neki od kriterija zaustavljanja

U ABC algoritmu, položaj izvora hrane predstavlja moguće rješenje problema optimizacije, a količina nektra izvora hrane odgovara kvaliteti povezanog rješenja. Broj zaposlenih pčela i pčela promatrača jednak je broju rješenja u populaciji. U koraku inicijalizacije, ABC algoritam generira nasumičnu distribuiranu početnu populaciju P ($C = 0$) od SN rješenja (položaja izvora hrane), gdje SN označava veličinu zaposlenih pčela ili pčela promatrača. Svako rješenje x_i ($i = 1, 2, ... SN$) je vektor dimenzija D. Ovdje, D je broj parametara optimizacije (broj nezavisnih varijabli). Nakon inicijalizacije, populacija položaja (rješenja) podliježe ciklusima ponavljanja ($C = 1, 2, ... MCN$) procesa pretraživanja zaposlenih pčela, pčela promatrača i pčela izviđača. Ciklus ponavljanja započinje tako što zaposlena pčela stvara izmjene informacija o položaja izvora hrane u svojoj memoriji, ovisno o lokalnim informacijama i ispitivanju količine nektra novih izvora hrane (rješenja). Ukoliko je količina nektra novog izvora viša od onog prethodnog, pčela memorira novi položaj i zaboravlja stari. Inače, zadržava u svojoj memoriji položaj prethodnog izvora. Za izmjenu starog rješenja u memoriji umjetne pčele, ABC algoritam koristi sljedeći izraz:

$$v_{\mathrm{i,j}} = x_{\mathrm{i,j}} + \phi_{\mathrm{i,j}} \cdot (x_{\mathrm{i,j}} - x_{\mathrm{k,j}}) \tag{16}$$

gdje je $k \in \{1, 2, \ldots SN\}$ i $j \in \{1, 2, \ldots D\}$ su nasumično odabrani indeksi. Iako je indeks k izabran nasumično, on mora biti različit od i. $\phi_{\mathrm{i,j}}$ je nasumičan broj između [-1, 1]. Njime se kontrolira proizvodnja susjednih izvora hrane oko $x_{\mathrm{i,j}}$ i predstavlja usporedbu dvaju položaja izvora hrane. Nakon što sve zaposlene pčele završe proces pretraživanja, one izmjenjuju informacije o količini nektra i položaju izvora hrane s pčelama promatračima i tu završava prvi korak ciklusa te započinje drugi. Drugi korak započinje tako što pčela promatrač procjenjuje podatke o izvoru hrane distribuiranih od strane zaposlenih pčela na plesnom području i odabire izvor hrane s većom količinom nektra i određenom vjerojatnošću p_{i}. Vrijednost vjerojatnosti p_{i} određuje se prema sljedećoj formuli:

$$p_{\mathrm{i}} = \frac{fit_{\mathrm{i}}}{\sum_{n=1}^{SN} fit_n} \tag{17}$$

gdje je fit_{i} vrijednost funkcije cilja položaja i koja je proporcionalna iznosu količine nektra na položaju i, a SN je broj izvora hrane koji je jednak broju zaposlenih pčela ili pčela promatrača. Kao i u slučaju zaposlene pčele, ona stvara izmjene o položaju u memoriji te provjerava količinu nektra izvora hrane. Ukoliko je količina nektra viša od one prethodne, pčela memorira novi položaj i zaboravlja stari. Inače, zadržava u svojoj memoriji položaj prethodnog izvora. U zadnjem koraku ciklusa, kada je nektar izvora hrane napušten od strane pčela, novi izvor hrane je nasumično odabran i zamijenjen s napuštenim. U ABC algoritmu, to je simulirano stvaranjem nasumičnog novog položaja koji zamjenjuje napušteni. Ako položaj ne može biti poboljšan kroz unaprijed određeni broj ciklusa, onda se pretpostavlja da je taj izvor hrane napušten. Vrijednost unaprijed određenog broja ciklusa je važan kontrolni parametar ABC algoritma te se naziva parametrom napuštanja L izvora hrane. Ukoliko izvor hrane nije poboljšan u $C \leq L$ ciklusa, taj izvor hrane se zamjenjuje s nasumičnim prema sljedećoj jednadžbi:

$$x_{\mathrm{i,j}} = lb_{\mathrm{j}} + unifrnd(0, 1) \cdot (ub_{\mathrm{j}} - lb_{\mathrm{j}}) \tag{18}$$

gdje je unifrnd(0, 1) nasumični broj između 0 i 1 prema uniformnoj distribuciji, a lb i ub donja i gornja granica nezavisnih varijabli. Ova tri koraka se ponavljaju kroz prethodno određeni broj ciklusa (maksimalan broj ciklusa MCN) ili dok se ne zadovolji neki od kriterija zaustavljanja.

5. IMPLEMENTACIJA

U ovome poglavlju će se pomoću tri različite metaheurističke metode pokušati pronaći optimalne vrijednosti faktora geometrijskog povećanja i napona izvora rendgenskih zraka, a držeći se zadanih ograničenja s ciljem minimiziranja odstupanja CT (eng. CT - *Computed Tomography*) vrijednosti promjera vanjskog cilindra od referentne vrijednosti. Odabrane metaheurističke metode su: metoda genetskog algoritma, optimizacija rojem čestice te optimizacija rojem pčela (ABC algoritam). Za početak, u ovome poglavlju, dan je kratki uvod o tome što je računalna tomografija. Zatim, primjena svake od prethodno navedenih triju metoda na konkretan problem optimizacije koristeći se softverskim paketom Matlab - *trial version* te na kraju usporedba dobivenih rezultata metoda.

5.1. Računalna tomografija

Općenito, računalna tomografija, poznatija kao CT, je dijagnostički medicinski postupak, koji poput tradicionalnih rendgenskih zraka, stvara višestruke fotografije ili snimke poprečnog presjeka unutrašnjosti tijela [26]. Fotografije poprečnog presjeka dobivene tijekom CT skeniranja se mogu preoblikovati u više ravnina, čime se može postići i trodimenzionalni prikaz unutrašnjosti tijela. Koristeći specijalnu opremu te stručnost za stvaranje i interpretaciju CT snimaka tijela, radiolozi mogu lakše dijagnosticirati zdravstvene probleme kao što su rak, kardiovaskularne bolesti, zarazne bolesti, upala slijepog crijeva, traume te mišićne i koštane poremećaje.

Industrijska računalna tomografija (eng. ICT - *Industrial Computed Tomography*) je bilo koji tomografski postupak potpomognut računalom, obično se radi o računalnoj tomografiji rendgenskim zrakama, koji koristi zračenje za proizvodnju trodimenzionalnih prikaza skeniranog objekta [27]. ICT je tehnika koja se koristi u mnogim područjima industrije za unutarnji pregled komponenti. ICT skeniranje koristi se kod otkrivanja grešaka, analizi kvara, u mjeriteljstvu, analizi montaže i u primjenama povratnog inženjerstva (eng. RE - *Reverse Engineering*).

Postoje dvije vrste skenera koji se koriste kod skeniranja objekata, a to su: linijski skeneri i konusni skeneri. Linijski skeneri su prva generacija ICT skenera. Kod njih rendgenske zrake koje se proizvode tvore linijski snop zraka. Taj snop zraka se zatim pošalje kroz objekt skeniranja, a dobiveni podaci se prikupljaju detektorom. Ti prikupljeni podaci se nakon toga rekonstruiraju za stvaranje trodimenzionalnih prikaza modela. Kod druge vrste skenera, objekt

skeniranja se postavlja na rotirajući stol. Kako se objekt skeniranja rotira snop rendgenskih zraka proizvodi dvodimenzionalne slike koje prikuplja detektor. Na kraju se te dvodimenzionalne slike obrađuju za stvaranje trodimenzionalnih prikaza iscrtavanjem vanjske i unutarnje geometrije objekta skeniranja.

5.1.1. Primjena ICT-a

Industrijska računalna tomografija najčešće se koristi u montažnoj ili vizualnoj analizi, usporedbi dijelova, otkrivanju oštećenja, šupljina, pukotina i kvara te za potrebe dimenzionalnih mjerenja te utvrđivanja odstupanja od oblika i položaja.

5.1.1.1. Montažna ili vizualna analiza te dimenzionalna mjerenja

Jedna od najpoznatijih oblika analize pomoću CT-a je montažna ili vizualna analiza. CT se u velikoj mjeri koristi u medicinske svrhe, kao alat za nadopunu ultrazvuku i rendgenu, za dijagnozu bolesti, ali i za preventivnu medicinu. Za industrijsko CT skeniranje sposobnost da se vidi unutrašnjost komponente je korisno jer se unutrašnji dijelovi vide u svom funkcionalnom položaju. Također, komponente se mogu vidjeti i analizirati bez rastavljanja. Neki softverski programi ICT skeniranja omogućuju da mjere budu uzete iz podataka skeniranja. Ova mjerenja su korisna za određivanje razmaka između montiranih dijelova ili za određivanje dimenzija pojedinog dijela.

5.1.1.2. Usporedba dijelova

Na današnjem tržištu dijelovi se mogu proizvesti u cijelom svijetu: dizajnirani u jednoj zemlji, proizvedeni u drugoj i sastavljeni u trećoj. Provjera da se neki dio poklapa s originalnim CAD (eng. CAD - *Computer Aided Design*) modelom je izuzetno važna, naročito ako je taj dio namijenjen za montažu. ICT omogućuje usporedbu dijelova ili dijelova prema CAD modelu. Odstupanja od unutarnjih i vanjskih geometrija mogu se prikazati na karti površinskih boja u trodimenzionalnom ili dvodimenzionalnom prikazu. Ovaj postupak je naročito koristan kod usporedbe istih dijelova od različitih dobavljača.

5.1.1.3. Otkrivanje oštećenja, šupljina, pukotina i kvara

Tradicionalno, otkrivanje oštećenja, šupljina, pukotina i kvara unutar objekta zahtijeva razaranje. No, CT otkriva unutarnje nedostatke bez razaranja dijela. Te nedostatke prikazuje u trodimenzionalnom prikazu dijela. ICT se koristi za otkrivanje nedostataka, kao što je poroznost

ili pukotina. U nekim softverskim programima poroznost je kategorizirana po boji gdje svaka boja odgovara određenom razredu veličina detektiranih pukotina.

5.2. Implementacija heurističkih metoda kod upotrebe metode računalne tomografije

Problem optimizacije sastoji se od pronalaženja optimalne vrijednosti funkcije cilja, pronalaženja minimalnog odstupanja CT vrijednosti promjera vanjskog cilindra od referentne vrijednosti, imajući u vidu dana ograničenja, domenu faktora geometrijskog povećanja i domenu napona izvora rendgenskih zraka. Jednadžba (19) prikazuje jednadžbu funkcije cilja.

$$\Delta d_{\mathrm{v}} = |\, 1{,}2451 - 0{,}4241 \cdot m - 0{,}0126 \cdot U + 4{,}4 \cdot 10^{3} \cdot m \cdot U \,| \rightarrow \min \tag{19}$$

Gdje je:

- Δd_{v}, mm - odstupanje CT vrijednosti promjera vanjskog cilindra od referentne vrijednosti
- m - faktor geometrijskog povećanja
- U, kV - napon izvora rendgenskih zraka

Odstupanje vrijednosti promjera vanjskog cilindra Δd_{v} izraženo je funkcijom ulaznih utjecajnih parametara na proces mjerenja metodom računalne tomografije koja je dobivena korištenjem metode djelomičnog plana pokusa, a u kojem je analiziran utjecaj sedam ulaznih faktora na praćenu izlaznu veličinu promjera cilindra [28]. Dobivena funkcija definira zavisnost odstupanja promjera u odnosu na promatrane ulazne parametre faktora geometrijskog povećanja i napona u slučaju za odabrani *Beam hardening filter* 1 - određivanje granice predmeta i pozadine primjenom automatskog odabira. Varijable u funkciji cilja su faktor geometrijskog povećanja m s domenom [2,19; 2,81] te napon izvora rendgenskih zraka U s domenom [45 kV; 50 kV].

5.2.1. Implementacija GA metode

Za implementaciju svih metoda koristit će se softver Matlab - *trial version*. Softver nudi dva načina provedbe metode genetskog algoritma. Prvi način je ručnim unosom redova koda u prozor Matlaba, čime se pozivaju već unaprijed izrađene funkcije metode GA koji dolaze u sklopu Matlab *toolboxa*. Drugi način je korištenjem već gotove aplikacije. Oba načina rade na istom principu zbog toga što koriste već unaprijed izrađeni kod funkcija, samo što kod drugog načina izgrađeno je cijelo grafičko sučelje te je stoga jednostavnije za upotrebu. U nastavku je prikazan osnovni kod za rješavanje konkretnog problema optimizacije putem metode GA.

Kod 5.1 Osnovni kod GA algoritma

```
01  clc;
02  clear;
03  close all;
04
05  ObjectiveFunction = @funkcija_cilja;
06  nvars = 2;
07  lb = [2.19 45];
08  ub = [2.81 50];
09  [x,fval] = ga(ObjectiveFunction,nvars,[],[],[],[],lb,ub);
10  fprintf('Najbolje pronađeno rješenje funkcije cilja : %g\n', fval);
```

Peti red koda odnosi se na pozivanje datoteke koja sadrži jednadžbu funkcije cilja. U ovom slučaju radi se o datoteci *funkcija_cilja.m*. Šesti red koda predstavlja broj nezavisnih varijabli pretraživanja. Sedmi i osmi red koda prikazuju donju i gornju granicu domene varijabli. U devetom redu, dio koda prije znaka jednakosti odnosi se na parametre koje će program ispisati na ekranu nakon završetka pretraživanja, a dio koda nakon znaka jednakosti odnosi se na ulazne parametre. Ulazni parametri koda su:

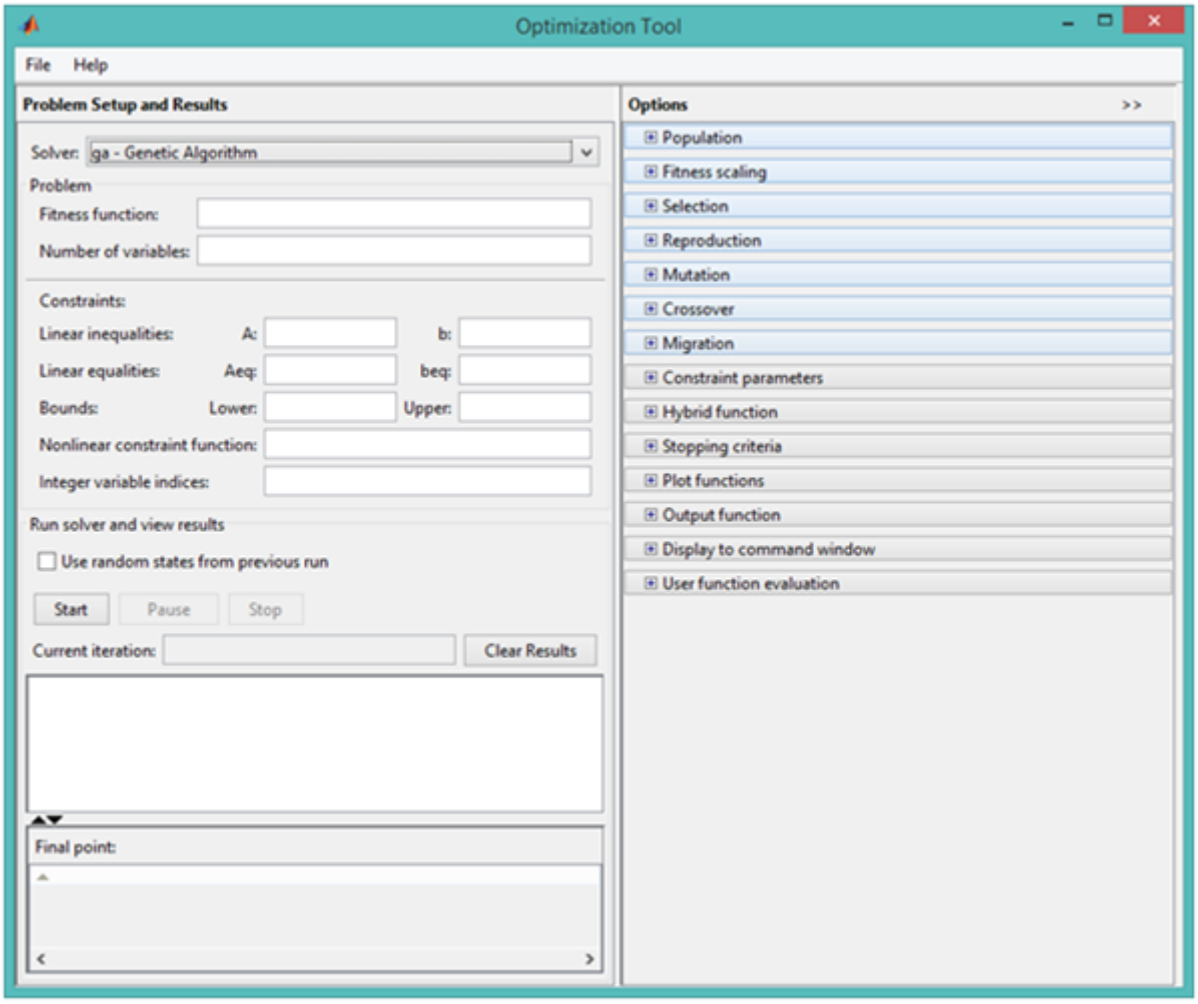
- *ObjectiveFunction* - odnosi se na pozivanje jednadžbe funkcije cilja
- *nvars* - broj nezavisnih varijabli pretraživanja
- [] - umjesto njih unose se vrijednosti *A*, *b*, *Aeq* i *Beq* koje treba definirati (njihovo značenje je definirano u daljnjem tekstu)
- *lb* - donja granica vrijednosti varijabli (ograničenje)
- *ub* - gornja granica vrijednosti varijabli (ograničenje)

Izlazni parametri koda su:

- *x* - koordinate u kojoj je postignuta najbolja funkcija cilja
- *fval* - vrijednost najbolje postignute funkcije cilja

Zadnji red koda služi samo za bolje prikazivanje dobivenog rješenja.

Za rješavanje ovog problema optimizacije koristit će se drugi način rješavanja putem aplikacije, korištenjem grafičkog sučelja, zbog jednostavnijeg reguliranja parametara genetičkog algoritma, kao što je populacija, selekcija, mutacija, reprodukcija ili kriterij zaustavljanja. Aplikacija se pokreće upisom *optimtool* u komandni prozor ili pritiskom na izbornik *Apps* > *Optimization* u izborniku Matlaba. Slika 5.1 prikazuje grafičko sučelje GA aplikacije s praznim poljima unosa.



Slika 5.1 Grafičko sučelje metode GA

Prozor aplikacije podijeljen je na dva dijela:

- na lijevi dio (*Problem Setup and Results)* - dio za unos osnovnih parametara problema (funkcije cilja i ograničenja) i prikaz rezultata
- te na desni dio (*Options*) - dio za podešavanje parametara metode GA.

Kako bi se mogao koristiti GA optimizacijski alat, najprije treba unijeti obvezne informacije u dio za unos osnovnih parametara problema. Unose se:

- *Fitness function* - Ovdje se unosi funkcija cilja koja se želi minimizirati. Radi se o pozivanju datoteke koja sadrži jednadžbu funkcije cilja. Unosi se u obliku *@funkcija_cilja*, gdje je *funkcija_cilja.m* datoteka m-formata koja vraća skalarnu vrijednost.
- *Number of variables* - Unosi se broj nezavisnih varijabli. Ovaj broj se mora poklapati s brojem varijabli u funkciji cilja.

- *Coinstraints* - Ovdje se unose ograničenja varijabli, kao što su:
    - Linearne nejednakosti (*Linear inequalities*) oblika $A \times x \leq b$, gdje je A matrica, a b vektor.
    - Linearne jednakosti (*Linear equalities*) oblika $Aeq \times x = beq$, gdje *Aeq* matrica, a *beq* vektor.
    - Domena nezavisnih varijabli (*Lower*/*Upper bounds*) koja se unosi u obliku vektora.
    - Funkcija nelinearnog ograničenja (*Nonlinear constraint function*) koja se unosi u obliku *@nelinearno_ogr*, gdje je *nelinearno_ogr.m* datoteka m-formata koja vraća vektore *c* i *ceq*. Nelinearne jednakosti su u formi $ceq = 0$, a nelinearne nejednakosti su u formi $c \leq 0$.
    - Indeksi cjelobrojnih varijabli (*Integer variables indices*) je vektor koji daje cjelobrojne vrijednost komponente *x*. Ukoliko je ovo polje ispunjeno, polja *Aeq* i *beq* moraju ostati neispunjena, sve funkcije nelinearnog ograničenja moraju vratiti prazno polje za *ceq* te vrsta populacije mora biti dvostruki vektor.

U desnom dijelu GA optimizacijskog alata podešavaju se različiti parametri koji su važni za izvođenje GA metode. Isto tako, ovdje se mogu definirati i načini prikazivanja rezultata. Za nas najzanimljiviji parametri koji se ovdje mogu podešavati su:

- *Population* - U ovom izborniku detaljno se definiraju svi parametre koji imaju veze s populacijom. Odabiru se vrsta populacije i način kreiranja funkcije, dok veličina populacije, polazna populacija, početni iznos i početni raspon mogu se definirati prema potrebama ili ostaviti na već zadanim vrijednostima (*default*).
- *Fitness scaling* - Njome se određuje funkcija skaliranja koja obavlja skaliranje. Skaliranje je pretvaranje „sirovog“ rezultata funkcije cilja u raspon vrijednosti rezultata koji je pogodan za odabir.
- *Selection* - Selektivna funkcija odabire roditelje za sljedeću generaciju na temelju svojih preračunatih vrijednosti iz funkcije skaliranja.
- *Reproduction* - Njime se određuje kako genetski algoritam stvara djecu sa svakom novom generacijom.

- *Mutation* - Mutacijske funkcije rade male slučajne promjene na pojedincima u populaciji, koje pružaju genetsku raznolikost i tako omogućiti genetskom algoritmu da pretražuje šire područje.
- *Crossover* - On kombinira dva pojedinca, ili roditelja, da formira novu osobu, ili dijete, za sljedeću generaciju.
- *Migration* - Migracija je kretanje pojedinaca između subpopulacija koje algoritam stvara ako se postavi da je veličina populacije vektor duljine veći od jedan. Time se postiže da najbolji pojedinci iz jedne subpopulacije zamjenjuju najgore pojedince iz druge.
- *Stopping criteria* - Njime se definira kriterij zaustavljanja, odnosno ono što uzrokuje zaustavljanje algoritma. To može biti:
    - *Generations* - Maksimalan broj iteracija što će ih algoritam izvršiti.
    - *Time limit* - Maksimalno vrijeme trajanja algoritma u sekundama.
    - *Fitness limit* - Ako vrijednost funkcije cilja premašuje ili je jednaka limitu funkcije cilja, algoritam se zaustavlja.
    - *Stall generations* - Algoritam se zaustavlja ukoliko je prosječna vrijednost promjene funkcije cilja manja od tolerancije.
    - *Stall time limit* - Ako nema poboljšanja vrijednosti funkcije cilja za neki interval vremena (u sekundama), algoritam se zaustavlja.
    - *Function tolerance* - Algoritam se zaustavlja ukoliko je prosječna vrijednost promjene funkcije cilja manja od tolerancije.
- *Plot function* - Tijekom generiranja rješenja grafički prikazuje razne korake u GA-u po određenom intervalu.
- *Display on command window* - Ovom opcijom određuje se količina informacija koja će se prikazati u Matlab komandnom prozoru kada se pokrene algoritam. Može se izabrati jedna od sljedećih opcija:
    - *Off* - Bez prikaza izlaza.
    - *Iterative* - Prikaz informacija pri svakoj iteraciji algoritma.
    - *Diagnose* - Prikaz informacija pri svakoj iteraciji. Osim toga, prikazuje informacije i mogućnosti koje su promijenjene od zadane vrijednosti.
    - *Final* - Prikazuje samo razlog zaustavljanja algoritma.

5.2.1.1. Optimiranje parametara funkcije cilja koristeći GA alat

Kako bi se mogao koristiti GA optimizacijski alat, najprije je potrebno napisati m-datoteku funkcije cilja. Kod 5.2 prikazuje funkciju cilja zapisanu u datoteci *funkcija_cilja.m*.

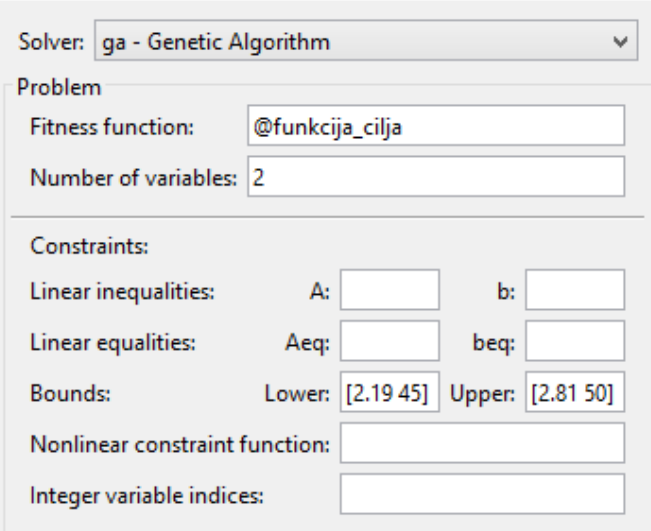
Kod 5.2 Funkcija cilja zapisana u datoteci m-formata

```
01  function y = funkcija_cilja(x)
02  y = abs(1.2451-0.4241*x(1)-0.0126*x(2)+4.4*x(1)*x(2)*10^-3);
03  end
```

Promatrajući prethodnu sliku mogu se vidjeti tri nepoznanice: y, $x(1)$ i $x(2)$. Zapravo, ne radi se o nepoznanicama, nego o simbolima koji zamjenjuju parametre funkcije cilja. U jednadžbi funkcije cilja:

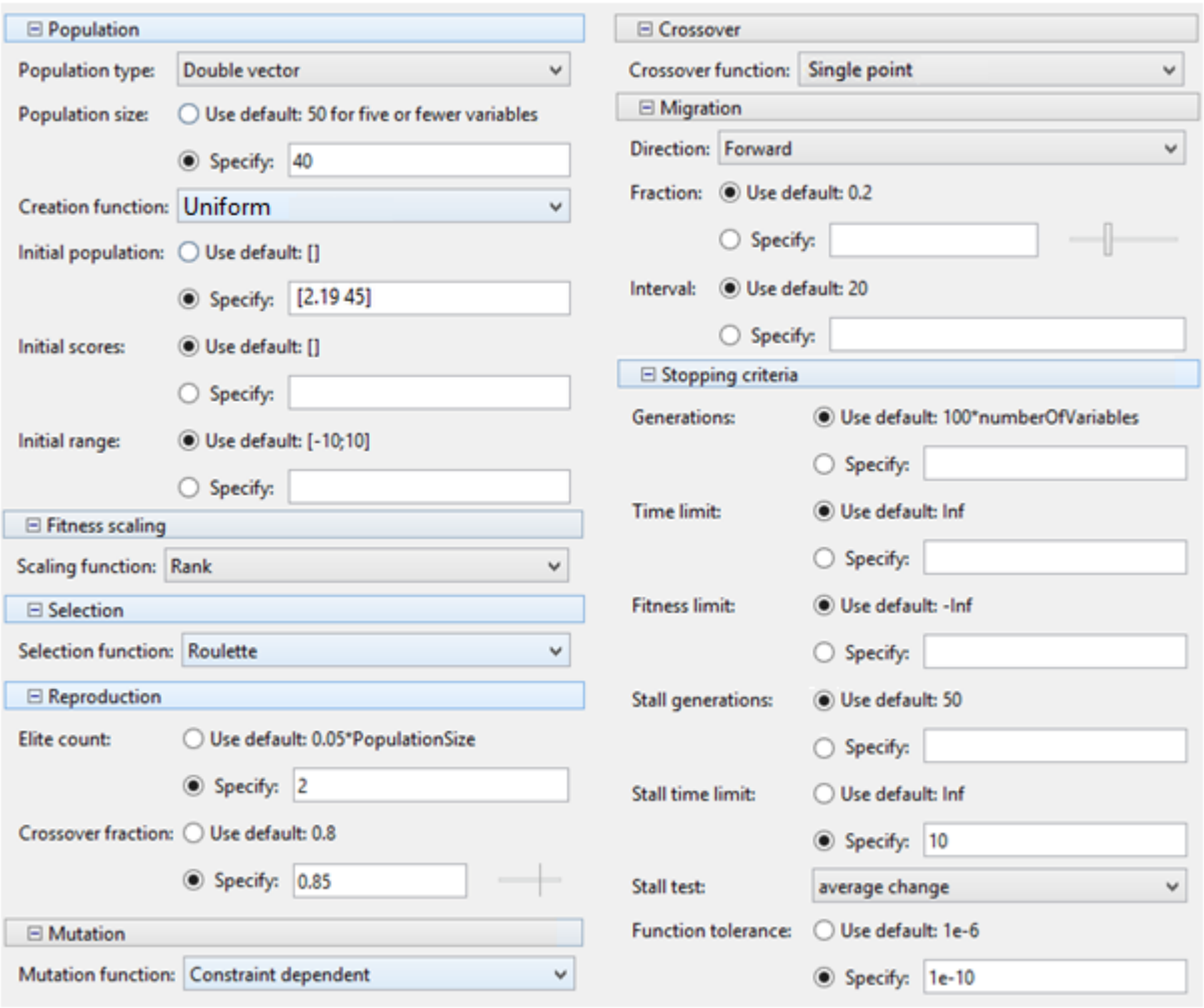
- y zamjenjuje Δd_v (odstupanje CT vrijednosti promjera vanjskog cilindra od referentne vrijednosti)
- $x(1)$ zamjenjuje m (faktor geometrijskog povećanja)
- dok $x(2)$ zamjenjuje U (napon izvora rendgenskih zraka)

Nakon što je definirana jednadžba funkcije cilja u datoteci *funkcija_cilja.m*, potrebno je s lijeve strane GA alata u polje *Fitness fuction* upisati ime m-datoteke u kojoj je definirana funkcija cilja (upisuje se *@funkcija_cilja*). Zatim, u polje *Number of variables* upisati vrijednost dva jer su dvije nezavisne varijable (m i U). Ograničenja u pogledu gornje i donje granice vrijednosti nezavisnih varijabli unose se u vektorskoj formi, dok ostala polja ograničenja ostaju prazna jer ne postoje druga poznata ograničenja. Sljedeća slika (Slika 5.2) prikazuje sve ovo što je ovdje rečeno.



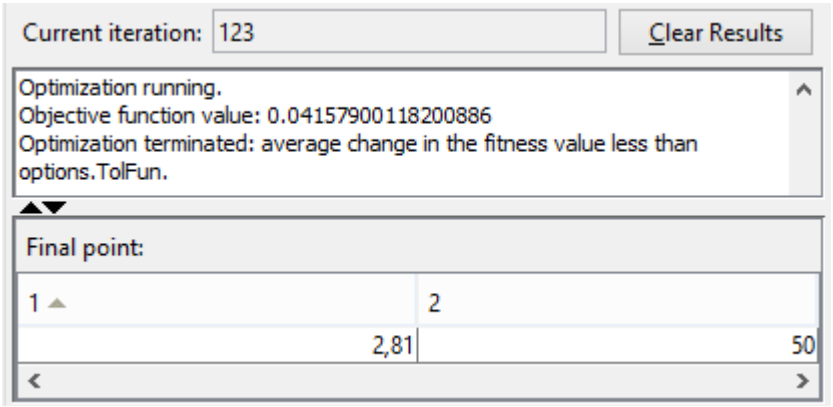
Slika 5.2 Unos funkcije cilja i ograničenja

Nakon unosa imena m-datoteke, u kojoj je definirana funkcija cilja, i definiranja ograničenja u pogledu gornje i donje granice vrijednosti nezavisnih varijabli, potrebno je podesiti GA alat prema određenim postavkama. Postavke GA alata prikazane su na slici 5.3. Za vrstu populacije koristi se vektorski prikaz, veličina populacije je 40, a način kreiranja funkcije je uniforman. Uniformno kreiranje funkcije stvara nasumičnu početnu populaciju s uniformnom raspodjelom unutar bilo kojih granica. Početna populacija postavljena je na donje vrijednosti granica varijabli *m* i *U*. Početni iznos i početni raspon populacije je postavljen na *default*. Početni iznos odnosi se na iznos funkcije cilja definirane početne populacije, a odabirom *default* algoritam izvrši izračun samostalno koristeći jednadžbu funkcije cilja. Skaliranje funkcije je po rangu. Tom opcijom skaliranje se vrši prema rangu svakog pojedinca. Rang pojedinca je njegova pozicija u razvrstanim rezultatima. Rang najjačeg pojedinca je 1, sljedećeg najjačeg 2, sljedećeg 3 i tako dalje. Ovim načinom skaliranja uklanja se efekt širenja sirovih rezultata. Operator selekcije je kolo sreće (Vidi poglavlje *4.1.2.1. Kolo sreće*). Kod reprodukcije odabrano je da se pamte dvije najbolje vrijednosti u svakom koraku, a vjerojatnost križanja je 85 %. Mutacija je postavljena da se prilagodi ograničenjima. Križanje je postavljeno na opciju jedna točka. Tom opcijom odabire se slučajan cijeli broj *n* između broja 1 i broja nezavisnih varijabli (ovdje je to broj 2), zatim se selektira vektorski genski zapis prvog roditelja koji je manji ili jednak od *n* i vektorski genski zapis drugog roditelja koji je veći od *n* čime se formira dijete prekriženih tih dvaju roditeljskih gena. Kod postavki migracije odabran je smjer naprijed, frakcija 0,2 te interval 20. Odabirom migracije u smjeru naprijed, migracija se odvija prema posljednjoj subpopulaciji, tj. pojedinci iz *n*-te subpopulacije prelaze u $n + 1$ subpopulaciju. Frakcijom se kontrolira broj pojedinaca koji se kreću između subpopulacija. Ako se pojedinci sele iz subpopulacije od 50 pojedinaca u subpopulaciju od 100 pojedinaca, a iznos frakcije je 0,2, deset osoba će migrirati ($0,2 \times 50$). Intervalom migracije kontrolira se broj iteracija koji mora proći između svake skupine migracija. Kriteriji zaustavljanja su: maksimalan broj iteracija 200, maksimalan broj iteracija bez poboljšanja funkcije cilja u odnosu na toleranciju funkcije cilja 50, vremensko ograničenje bez poboljšanja funkcije cilja 10 sekundi te iznos tolerancije funkcije cilja 10^{-10}.



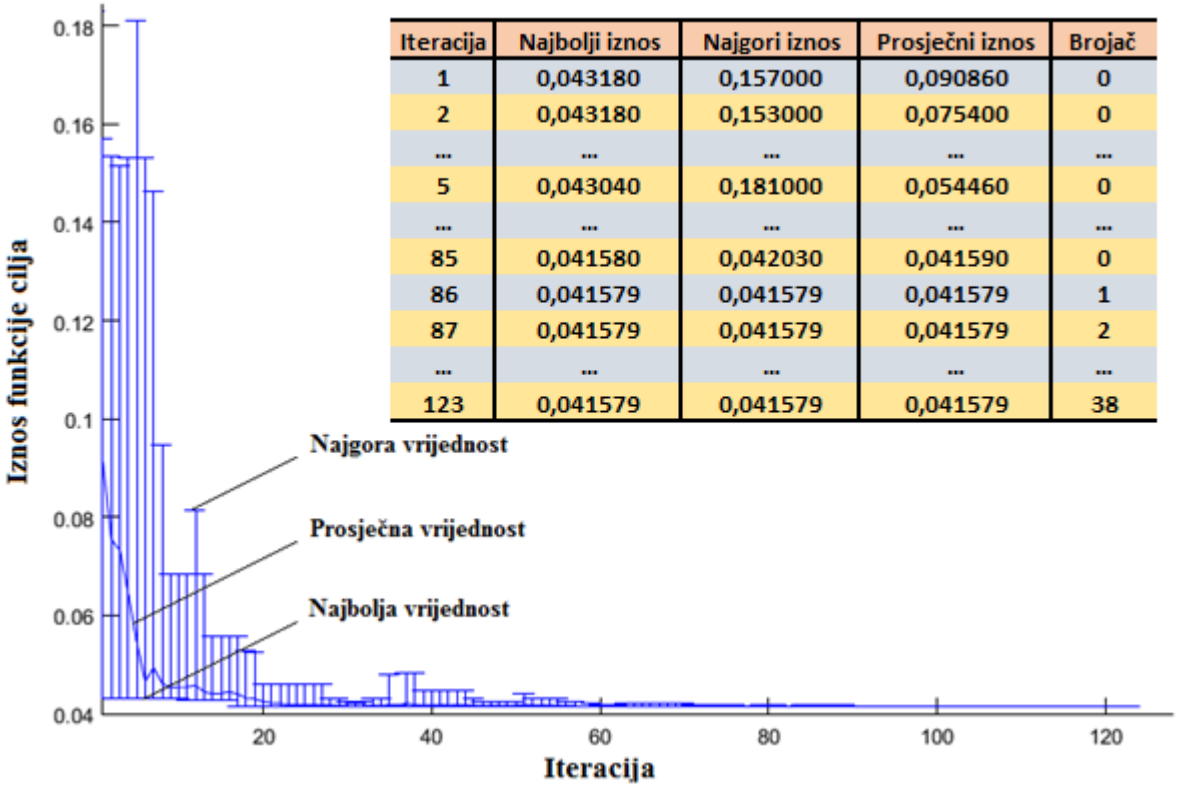
Slika 5.3 Postavke genetskog algoritma

Nakon pritiska tipke *Start* započinje proces optimiranja koristeći genetski algoritam. Nakon nekog vremena, ovisno o snazi i brzini računala, genetski algoritam je odredio vrijednosti nezavisnih varijabli te iznos funkcije cilja. Dobiveni rezultati prikazani su na slici 5.4.



Slika 5.4 Rezultati dobiveni GA metodom

Nakon 123. iteracije algoritam je prestao s pretraživanjem pošto je prosječna promjena iznosa funkcije cilja manja od zadane vrijednosti (10^{-10}). Konačni iznos funkcije cilja je Δd_v = 0,041579 mm za iznos varijabli m = 2,81 i U = 50 kV. Sljedeća slika (Slika 5.5) prikazuje kretanje iznosa funkcije cilja iz iteracije u iteraciju.



Iteracija	Najbolji iznos	Najgori iznos	Prosječni iznos	Brojač
1	0,043180	0,157000	0,090860	0
2	0,043180	0,153000	0,075400	0
...	...	...	...	...
5	0,043040	0,181000	0,054460	0
...	...	...	...	...
85	0,041580	0,042030	0,041590	0
86	0,041579	0,041579	0,041579	1
87	0,041579	0,041579	0,041579	2
...	...	...	...	...
123	0,041579	0,041579	0,041579	38

Slika 5.5 Promjena iznosa funkcije cilja GA metodom

Može se vidjeti kako iznos funkcije cilja varira kako algoritam napreduje. Na slici razlikujemo tri različita iznosa funkcije cilja, a to su: najbolja vrijednost, najgora vrijednost i prosječna vrijednost funkcije cilja za svaku iteraciju. Na početku procesa iznos najgore vrijednosti funkcije cilja iznosi 0,157000 mm da bi na 5-toj iteraciji postigao svoju najvišu vrijednost od 0,181000 mm. Nakon svoje najviše vrijednosti, iznos najgore vrijednosti funkcije cilja opada sa svakom sljedećom iteracijom. Od samog početka najbolja vrijednost funkcije cilja je vrlo blizu optimalnom rješenju te na već 20-toj iteraciji postiže svoj optimum. Prosječna vrijednost funkcije cilja kreće s iznosom 0,090860 mm i dalje strmovito opada. Na 86-toj iteraciji iznosi najgore, najbolje i prosječne vrijednosti funkcije cilja se podudaraju i tako ostaje do samoga kraja, odnosno dok se nije zadovoljio neki od kriterija zaustavljanja. Brojač služi za brojanje podudaranja iznosa prosječne i najbolje vrijednosti funkcije cilja. Brojač ima vrijednost nula na početku pa svaki put kada naiđe na podudaranje iznosa prosječne i najbolje

vrijednosti funkcije cilja doda mu se 1. Ukoliko naiđe na različiti iznos prosječne i najbolje vrijednosti funkcije cilja resetira se na 0 i proces brojanja počinje ispočetka.

5.2.2. Implementacija PSO metode

Metoda optimizacije rojem čestica (PSO) je implementirana kroz softver Matlab. U Matlabu već postoji naredba za prizivanje PSO algoritma. No, kako se iza tog algoritma nalazi kod od 700-tinjak redova, on je prilično zamršen i nepregledan te zbog toga je izrađen vlastiti kod za PSO metodu. Kod algoritma je prikazan u nastavku (Kod 5.3), a njegovo objašnjenje je dano odmah nakon toga.

Kod 5.3 Kod PSO algoritma

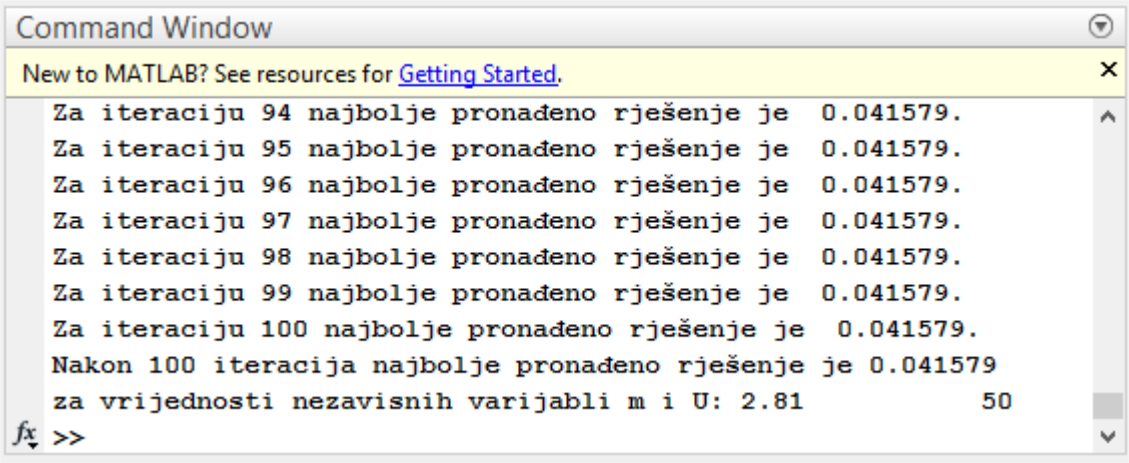
```
clc;
clear;
close all;

%% 1. Definicija problema
FunkcijaCilja = @funkcija_cilja;  % Funkcija cilja
nVar = 2;                         % Broj nezavisnih varijabli
VarSize = [1 nVar];               % Veličina matrice varijabli
lb = [2.19 45];                   % Donja granica nezavisnih varijabli
ub = [2.81 50];                   % Gornja granica nezavisnih varijabli

%% 2. Parametri algoritma
MaxIt = 100;                      % Maksimalni broj iteracija
nPop = 40;                        % Veličina populacije
w = 1;                            % Koeficijent inercije
c1 = 2;                           % Faktor kognitivnog učenja
c2 = 2;                           % Faktor socijalnog učenja
MaxBrzina = 0.2 * (ub - lb);      % Maksimalna brzina čestice
MinBrzina = - MaxBrzina;          % Minimalna brzina čestice

%% 3. Inicijalizacija
% Struktura prazne čestice
prazna_cestica.Lokacija = [];     % Lokacija čestice u prostoru
                                  % pretraživanja
prazna_cestica.Brzina = [];       % Brzina čestice
prazna_cestica.Cost = [];         % Vrijednost funkcije cilja čestice
prazna_cestica.Best.Lokacija = [];% Najbolja pronađena lokacija čestice
prazna_cestica.Best.Cost = [];    % Najbolja pronađena vrijednost
                                  % funkcije cilja
% Kreiranje populacije čestica
cestica = repmat(prazna_cestica, nPop, 1);     % Ponavljanje matrice
                                               % dimenzija 40x1
% Inicijalizacija globalno najboljeg rješenja
GlobalBest.Cost = inf;                % Početna vrijednost beskonačno
% Inicijalizacija članova populacije
for i = 1:nPop
    % Generiranje nasumičnih rješenja
    cestica(i).Lokacija = unifrnd(lb, ub, VarSize); % Uniformna
                          % nasumična distribucija veličine VarSize
    % Početna brzina
    cestica(i).Brzina = zeros(VarSize); % Matrica sa svim elementima 0
                                        % i veličine VarSize
```

```
43        % Evaluacija
44        cestica(i).Cost = FunkcijaCilja(cestica(i).Lokacija);
45        % Ažuriranje najboljeg rješenja
46        cestica(i).Best.Lokacija = cestica(i).Lokacija;
47        cestica(i).Best.Cost = cestica(i).Cost;
48        % Ažuriranje globalno najboljeg rješenja
49        if cestica(i).Best.Cost < GlobalBest.Cost
50            GlobalBest = cestica(i).Best;
51        end
52    end
53    % Najbolja vrijednost funkcije cilja pri svakoj iteraciji
54    BestCosts = zeros(MaxIt, 1);                % Prazna matrica 100x1
55
56    %% 4. Glavna petlja algoritma
57    for it = 1:MaxIt                            % 100 iteracija
58        for i = 1:nPop                          % 40 čestica
59            % Ažuriranje brzina
60            cestica(i).Brzina = w*cestica(i).Brzina + c1*rand(VarSize) ...
61            .* (cestica(i).Best.Lokacija - cestica(i).Lokacija) + c2* ...
62            rand(VarSize) .* (GlobalBest.Lokacija - cestica(i).Lokacija);
63            % Uzimanje u obzir MinBrzina i MaxBrzina
64            cestica(i).Brzina = max(cestica(i).Brzina, MinBrzina);
65            cestica(i).Brzina = min(cestica(i).Brzina, MaxBrzina);
66            % Ažuriranje lokacija
67            cestica(i).Lokacija = cestica(i).Lokacija + cestica(i).Brzina;
68            % Uzimanje u obzir lb i ub pri određivanju lokacija
69            cestica(i).Lokacija = max(cestica(i).Lokacija, lb);
70            cestica(i).Lokacija = min(cestica(i).Lokacija, ub);
71            % Evaluacija
72            cestica(i).Cost = FunkcijaCilja(cestica(i).Lokacija);
73            % Ažuriranje najboljeg rješenja
74            if cestica(i).Cost < cestica(i).Best.Cost
75                cestica(i).Best.Lokacija = cestica(i).Lokacija;
76                cestica(i).Best.Cost = cestica(i).Cost;
77                % Ažuriranje globalno najboljeg rješenja
78                if cestica(i).Best.Cost < GlobalBest.Cost
79                    GlobalBest = cestica(i).Best;
80                end
81            end
82        end
83    % Spremanje globalno najboljeg rješenja
84    BestCosts(it) = GlobalBest.Cost;
85    % Prikazivanje informacija po iteracijama
86    disp(['Za iteraciju ' num2str(it) ' najbolje pronađeno rješenje je '...
87    num2str(BestCosts(it)) ,'.']);
88    end
89
90    %% 5. Prikaz rezultata
91    disp(['Nakon 100 iteracija najbolje pronađeno rješenje je ' ...
92    num2str(GlobalBest.Cost)]);
93    disp(['za vrijednosti nezavisnih varijabli m i U: ' ...
94    num2str(GlobalBest.Lokacija)]);
95    plot(BestCosts, 'LineWidth', 2, 'color', 'red');
96    xlabel('Iteracija');
97    ylabel('Vrijednost funkcije cilja');
98    grid off;
```

Kod PSO algoritma sastoji se od 100-tinjak redova koda, gdje je kod podijeljen u pet dijelova: definicija problema, parametri algoritma, inicijalizacija, glavna petlja algoritma i na

prikaz rezultata. U prvom dijelu, unesena je naredba za pozivanje jednadžba funkcije cilja (poziva se datoteka *funkcija_cilja.m*) te broj (*nVar* = 2), veličina matrice (matrica 1 × 2) i donja (*lb*) i gornja (*ub*) granica nezavisnih varijabli. U drugom dijelu, uneseni su parametri: maksimalan broj iteracija (*MaxIt* = 100), veličina populacije čestica (*nPop* = 40), koeficijent inercije (*w* = 1), faktor kognitivnog (C_1 = 2) i socijalnog (C_2 = 2) učenja te maksimalna i minimalna brzina čestice koja je definirana u 18. i 19. redu koda. Važno je spomenuti da je ovdje *MaxIt* ujedno i kriterij zaustavljanja. U trećem dijelu, kreira se struktura prazne čestice i početna populacija čestica. Struktura čestice sastoji se od lokacije, brzine i vrijednosti funkcije cilja za danu lokaciju. Populacija čestica sastoji se od 40 čestica kojima je dana nasumična lokacija u prostoru rješenja držeći se danih granica u pogledu gornje i donje granice nezavisnih varijabli te je za svaku tu lokaciju izračunata vrijednost funkcije cilja. Brzina svih čestica populacije je na početku 0 zbog toga što su čestice još u fazi mirovanja. Također, u inicijalizaciji definiran je i parametar *GlobalBest* koji pamti do sada najbolje pronađeno rješenje. U njemu su sadržani parametar lokacije i vrijednost funkcije cilja. U fazi inicijalizacije radi se o najboljoj vrijednosti funkcije cilja čestice koja svojim položajem u prostoru rješenja daje minimalnu vrijednost funkcije cilja. Ta vrijednost se ažurira svakom sljedećom iteracijom, naravno, ukoliko dođe do poboljšanja vrijednosti funkcije cilja. U četvrtom dijelu, provodi se za svaku česticu 100 iteracija u nadi da se postigne globalni minimum. U svakoj iteraciji, za svaku česticu populacije, ažurira se brzina, lokacija te vrijednost funkcije cilja za danu lokaciju. Kako ovdje dolazi do ažuriranja lokacija i brzine čestica, potrebno je postaviti ograničenje da nova lokacija/brzina ne izlazi iz danih ograničenja. To je postignuto na način da se izvrši usporedba nove lokacije/brzine s ograničenjima. Usporedba brzine prikazana je u 64. i 65. redu koda, dok usporedba lokacije je prikazana u 69. i 70. redu koda. Nakon dobivenih novih lokacija čestica, računaju se nove vrijednosti funkcije cilja te se ažurira *GlobalBest* ukoliko je postignuto poboljšanje u odnosu na prethodne iteracije. Isto tako, u ovom dijelu koda definiran je parametar *BestCosts(it)*, koji nam omogućava prikazivanje najbolje vrijednosti funkcije cilja nakon svake iteracije. Na kraju, u petom dijelu koda, definirane su naredbe za prikazivanje globalno najboljeg rješenja, odnosno prikaz koordinata najbolje lokacije te vrijednost funkcije cilja za tu lokaciju. Osim toga, naredbom *plot* omogućen je grafički prikaz rezultata.

Nakon što su svi parametri definirani, pritiskom na tipku *Run* u glavnom izborniku Matlaba, PSO algoritam započinje sa pretraživanjem prostora rješenja. Nakon provedenih 100 iteracija sve čestice nalaze se na istoj lokaciji te iznosi njihovih funkcija cilja su isti. U nastavku je prikazano rješenje dobiveno napisanim algoritmom (Slika 5.6).

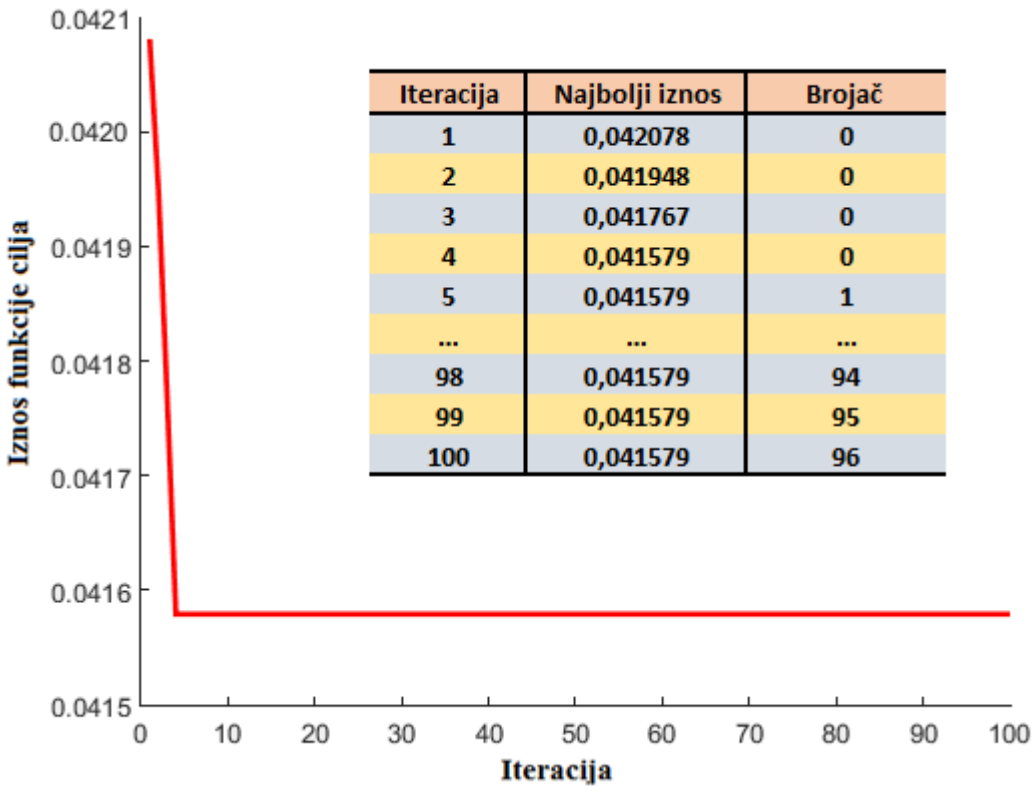


Slika 5.6 Rezultati dobiveni PSO metodom

Kao što se može vidjeti s prethodne slike, najbolje postignuto rješenje nakon 100 provedenih iteracija je Δd_v = 0,041579 mm, a koordinate (vrijednosti) nezavisnih varijabla su:

- m = 2,81
- U = 50 kV

Radi boljeg razumijevanja dobivenih rezultata u nastavku je dan grafički prikaz kretanja vrijednosti funkcije cilja iz iteracije u iteraciju (Slika 5.7).



Iteracija	Najbolji iznos	Brojač
1	0,042078	0
2	0,041948	0
3	0,041767	0
4	0,041579	0
5	0,041579	1
...	...	...
98	0,041579	94
99	0,041579	95
100	0,041579	96

Slika 5.7 Promjena iznosa funkcije cilja PSO metodom

S prethodnog grafa vidljivo je da funkcija cilja postiže svoju minimalnu vrijednost već nakon 4. iteracije i takva ostaje do samoga kraja. To je postignuto zbog veoma kvalitetno izrađenog algoritma, koji ne dopušta rasipanje. Da se još postavilo da je jedna od čestica u početku na lokaciji [2,81 50], a ne da su početne lokacije nasumično odabrane, imajući u vidu gornju i donju granicu varijabli *m* i *U*, algoritam bi pronašao globalni minimum već u samoj fazi inicijalizacije. Brojač služi za brojanje podudaranja dvaju iznosa funkcije cilja, prethodnog i sljedećeg iznosa. Kao što je već rečeno, iznos najbolje vrijednosti funkcije cilja ostaje nepromijenjen do samog kraja, što se može vidjeti preko brojača koji pokazuje broj 96. Znači, od 4. pa do 100. iteracije algoritam nije mogao pronaći bolje rješenje. Naravno, to je zbog danih ograničenja u pogledu gornje granice varijabli *m* i *U*.

5.2.3. Implementacija ABC metode

Metoda optimizacije rojem pčela je implementirana kroz softver Matlab. U Matlabu ne postoji naredba za pozivanje te metode niti bilo kakvo grafičko sučelje u kojem bi se ta metoda provela. Zbog toga je na temelju stečenog znanja izrađen algoritam. Ovdje je izrađen algoritam kolonije umjetnih pčela (ABC algoritam). Kod algoritma je prikazan u nastavku (Kod 5.4), a njegovo objašnjenje je dano odmah nakon toga.

Kod 5.4 Kod ABC algoritma

```
001 clc;
002 clear;
003 close all;
004
005 %% 1. Definicija problema
006 FunkcijaCilja = @funkcija_cilja;     % Funkcija cilja
007 nVar = 2;                            % Broj nezavisnih varijabli
008 VarSize = [1 nVar];                  % Veličina matrice varijabli
009 lb = [2.19 45];                      % Donja granica varijabli
010 ub = [2.81 50];                      % Gornja granica varijabli
011
012 %% 2. Parametri ABC-a
013 MaxIt = 100;                         % Maksimalni broj iteracija
014 nPop = 50;                           % Veličina populacije
015 nOnlooker = nPop;                    % Broj pčela promatrača
016 L = 25;                              % Parametar napuštanja
017
018 %% 3. Inicijalizacija
019 % Struktura prazne pčele
020 prazna_pcela.Lokacija = [];          % Lokacija pčele u prostoru
021 prazna_pcela.Cost = [];              % Vrijednost funkcije cilja
022 % Kreiranje populacije pčela
023 pcela = repmat(prazna_pcela, nPop, 1); % Ponavljanje matrice (50x1)
024 % Inicijalizacija globalno najboljeg rješenja
025 BestSol.Cost = inf;                    % Polazna vrijednost beskonačno
026 % Inicijalizacija članova populacije
027 for i = 1:nPop                        % i ide od 1 do 50
```

```
028      % Generiranje nasumičnih rješenja
029      pcela(i).Lokacija = unifrnd(lb, ub, VarSize);% Uniformna nasumična
030                                                   % distribucija
031      pcela(i).Cost = FunkcijaCilja(pcela(i).Lokacija);
032      if pcela(i).Cost <= BestSol.Cost
033          BestSol = pcela(i);
034      end
035  end
036  % Brojač nenapuštanja
037  C = zeros(nPop, 1);                    % Prazna matrica 50x1
038  % Najbolja vrijednost funkcije cilja pri svakoj iteraciji
039  BestCost = zeros(MaxIt, 1);            % Prazna matrica 100x1
040
041  %% 4. ABC glavna petlja
042  for it = 1:MaxIt
043      % Zaposlene pčele
044      for i = 1:nPop
045          K = [1:i-1 i+1:nPop];            % U matrici se briše stupac i
046          k = K(randi([1 numel(K)]));      % Nasumični k različit od i
047          phi = unifrnd(-1, +1, VarSize); % Koeficijent akceleracije
048          % Nova lokacija pčele
049          nova_pcela.Lokacija = pcela(i).Lokacija + phi .* ...
050          (pcela(i).Lokacija - pcela(k).Lokacija);
051          nova_pcela.Lokacija = max(nova_pcela.Lokacija, lb);
052          nova_pcela.Lokacija = min(nova_pcela.Lokacija, ub);
053          nova_pcela.Cost = FunkcijaCilja(nova_pcela.Lokacija);
054          % Usporedba
055         if nova_pcela.Cost <= pcela(i).Cost
056              pcela(i) = nova_pcela;
057          else
058              C(i) = C(i) + 1;
059          end
060      end
061      % Izračun vrijednosti vjerojatnosti odabira
062      F = zeros(nPop,1);                    % Matrica 50x1
063      for i = 1:nPop
064          F(i) = pcela(i).Cost;
065      end
066      P = F / sum(F);                       % Vjerojatnost odabira
067      % Pčele promatrači
068      for m = 1:nOnlooker
069          % Odabir izvora
070          i = RouletteWheelSelection(P);
071          % Odabir k
072          K = [1:i-1 i+1:nPop];
073          k = K(randi([1 numel(K)]));
074          phi = unifrnd(-1, +1, VarSize); % Koeficijent akceleracije
075          % Nova lokacija pčele
076          nova_pcela.Lokacija = pcela(i).Lokacija + phi .* ...
077          (pcela(i).Lokacija - pcela(k).Lokacija);
078          nova_pcela.Lokacija = max(nova_pcela.Lokacija, lb);
079          nova_pcela.Lokacija = min(nova_pcela.Lokacija, ub);
080          nova_pcela.Cost = FunkcijaCilja(nova_pcela.Lokacija);
081          % Usporedba
082          if nova_pcela.Cost <= pcela(i).Cost
083              pcela(i) = nova_pcela;
084          else
085              C(i) = C(i) + 1;
086          end
087      end
```

```
088      % Pčele izviđači
089      for i = 1:nPop
090          if C(i) >= L
091              pcela(i).Lokacija = lb+unifrnd(0,1,VarSize).*(ub - lb);
092              pcela(i).Cost = FunkcijaCilja(pcela(i).Lokacija);
093              C(i)=0;
094          end
095      end
096      % Ažuriranje najboljeg rješenja
097      for i=1:nPop
098          if pcela(i).Cost <= BestSol.Cost
099              BestSol = pcela(i);
100          end
101      end
102      % Spremanje globalno najboljeg rješenja
103      BestCost(it) = BestSol.Cost;
104      disp(['Za iteraciju ' num2str(it) 'najbolje rješenje je = ' ...
105      num2str(BestCost(it))]);
106  end
107
108  %% 5. Rezultati
109  disp(['Nakon 100 iteracija najbolje pronađeno rješenje je ' ...
110  num2str(BestSol.Cost)]);
111  disp(['za vrijednosti nezavisnih varijabli m i U: ' ...
112  num2str(BestSol.Lokacija)]);
113  plot(BestCost,'LineWidth', 2, 'color', 'red');
114  xlabel('Iteracija');
115  ylabel('Vrijednost funkcije cilja');
116  grid off;
```

Kod ABC algoritma sastoji se od 120-tak redova koda, gdje je kod podijeljen u pet dijelova: definicija problema, parametri ABC-a, inicijalizacija, ABC glavna petlja i na rezultate. U prvom dijelu koda, unesena je naredba za pozivanje jednadžba funkcije cilja (poziva se datoteka *funkcija_cilja.m*) te broj (*nVar* = 2), veličina matrice (matrica 1 × 2) te donja (*lb*) i gornja (*ub*) granica nezavisnih varijabli. U drugom dijelu koda, uneseni su parametri: maksimalan broj iteracija (*MaxIt* = 100), veličina populacije pčela (*nPop* = 50), broj pčela promatrača (50) te faktor napuštanja (*L* = 25).Važno je spomenuti da je ovdje *MaxIt* ujedno i kriterij zaustavljanja. U trećem dijelu, kreira se struktura prazne pčele i početna populacija pčela. Struktura pčele sastoji se od lokacije i vrijednosti funkcije cilja za danu lokaciju. Populacija pčela sastoji se od 50 pčela kojima je dana nasumična lokacija u prostoru rješenja držeći se danih granica u pogledu gornje i donje granice nezavisnih varijabli te je za svaku tu lokaciju izračunata vrijednost funkcije cilja. Također, u inicijalizaciji definirane su i varijable *BestCost* i *BestSol*. Varijabla *BestCost* pamti najbolju vrijednost funkcije cilja u svakoj iteraciji, dok *BestSol* pamti najbolje rješenje do sada pronađeno. U fazi inicijalizacije radi se o najboljoj vrijednosti funkcije cilja pčele, koja svojim položajem u prostoru rješenja daje minimalnu vrijednost funkcije cilja. Ta vrijednost se ažurira svakom sljedećom iteracijom, naravno, ukoliko dođe do poboljšanja vrijednosti funkcije cilja. Isto tako, pojavljuje se i varijabla *C* koja glumi brojač nenapuštanja.

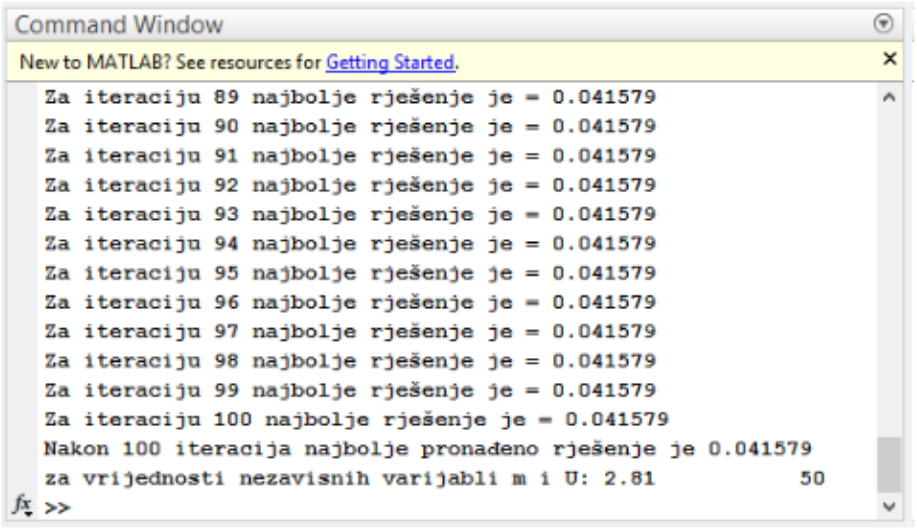
Radi se o brojanju slijednih iteracija u kojima pčele nisu promijenile prethodnu lokaciju. Ukoliko dođe do promijene lokacije, brojač se resetira na 0 i počinje brojati ispočetka. Četvrti dio koda (ABC glavna petlja) podijeljen je na tri koraka: na korak zaposlenih pčela, na korak pčela promatrača te na korak pčela izviđača. U koraku zaposlenih pčela, najprije je potrebno odrediti indeks *k* koji mora biti različit od *i*. Ukoliko bi *k* bio isti kao *i*, nova lokacija bi ostala ista prethodnoj (jednadžba (16)). Naravno, nova lokacija ne znači nužno i bolje rješenje pa je stoga potrebno usporediti vrijednosti funkcije cilja prethodne i nove lokacije. Ukoliko je vrijednost funkcije cilja za novu lokaciju manja u odnosu na vrijednost funkcije cilja stare lokacije, pčela prelazi na novu lokaciju. Inače, pčela ostaje na staroj lokaciji te se dodaje +1 brojaču nenapuštanja promatrajuće pčele. Na završetku tog koraka potrebno je izračunati vjerojatnost odabira *i*-te lokacije zaposlene pčele od strane pčela promatrača. Drugi korak započinje odabirom jednog od izvora hrane od strane pčela promatrača. To se čini uz pomoć koda za RWC selekciju (Kod 5.5) i vrijednosti vjerojatnosti (*P*) odabira *i*-te lokacije.

Kod 5.5 Kod za RWC selekciju

```
01  function i = KoloSrece(P)
02      r = rand;                       % Slučajni broj između 0 i 1
03      C = cumsum(P);                  % Kumulativna suma
04      i = find(r<=C, 1, 'first');
05  end
```

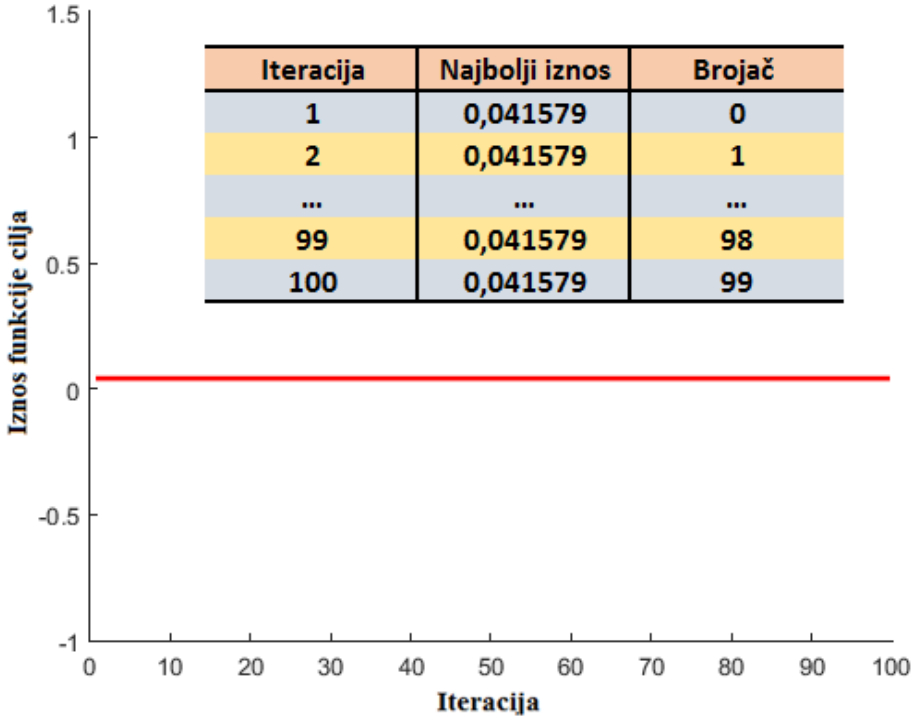
Nakon što su pčele promatrači odabrale jedan od mogućih izvora hrane, one postaju zaposlene. Postajući zaposlenom pčelom za nju se traži, kao i u koraku zaposlene pčele, nova lokacija. Pronalaskom nove lokacije vrši se usporedba nove i stare lokacije, tj. usporedba njihovih vrijednosti funkcije cilja. Pčela odabire lokaciju s manjom vrijednosti funkcije cilja. Ako je to stara lokacija, brojaču nenapuštanja se dodaje +1, a ako je to nova lokacija, brojač nenapuštanja se resetira na 0. Ovom usporedbom se završava drugi korak. Treći korak započinje usporedbom brojača nenapuštanja svih pčela s parametrom napuštanja (L = 25). Ukoliko je brojač nenapuštanja *i*-te pčele veći ili jednak od *L*, toj pčeli se pronalazi nova lokacija prema jednadžbi (18) te se računa vrijednost funkcije cilja za tu lokaciju. Nakon svega toga, vrši se ažuriranje globalno najboljeg rješenja (*BestSol*) te se spremaju najbolja pronađena rješenja po iteracijama (*BestCost*). U petom dijelu, kao i u prethodnom algoritmu, definirane su naredbe za prikazivanje globalno najboljeg rješenja na kraju pretraživanja, odnosno prikaz koordinata najbolje pronađene lokacije i vrijednosti funkcije cilja za tu lokaciju. Također, naredbom *plot* omogućen je grafički prikaz rezultata po iteracijama.

Nakon što su svi parametri definirani, pritiskom na tipku *Run* u glavnom izborniku Matlaba, ABC algoritam započinje sa pretraživanjem prostora rješenja. Nakon provedenih 100 iteracija dobiveni su sljedeći rezultati (Slika 5.8).



Slika 5.8 Rezultati dobiveni ABC algoritmom

Sa slike je vidljivo da nakon 100 iteracija najbolje postignuto rješenje je Δd_v = 0,041579 mm, a vrijednosti nezavisnih varijabla su: m = 2,81 i U = 50 kV. Pošto prethodna slika ne prikazuje sve najbolje vrijednosti funkcije cilja po iteracijama, dana je sljedeća slika (Slika 5.9).



Slika 5.9 Promjena iznosa funkcije cilja ABC algoritmom

S prethodnog grafa vidljivo je da funkcija cilja postiže svoju minimalnu vrijednost već nakon prve iteracije. To je postignuto jednim od sljedeća tri razloga:

- jedna ili više pčela u fazi inicijalizacije je dobila početnu lokaciju s koordinatama [2,19 50],
- izračunate nove vrijednosti koordinata lokacije jedne ili više pčela podudaraju se s koordinatama gornjih ograničenja
- ili su koordinate nove lokacije zamijenjene s vrijednostima gornjih granica varijabli u prvoj iteraciji, što je postignuto usporedbom izračunatih novih vrijednosti koordinata lokacija pčela i ograničenja u pogledu gornje granice varijabli u koraku zaposlenih pčela ili koraku pčela promatrača.

Malo je izgledno da je to postignuto u fazi inicijalizacije, zbog toga što je početna populacija pčela generirana nasumičnom uniformnom distribucijom. Isto tako, malo je izgledno da se radi o izračunatim novim koordinatama koje su jednake vrijednostima gornjih ograničenja. Stoga, zaključujemo da se najvjerojatnije radi o iznosima koordinata novih lokacija koje iskaču iz gornjih ograničenja te su stoga zamijenjene s gornjim granicama varijabli u fazi usporedbe, bilo da je to učinjeno u koraku zaposlenih pčela ili koraku pčela promatrača.

5.2.4. Usporedba rezultata svih metoda

Za usporedbu rezultata dobivenih svakom od metoda izrađena je sljedeća tablica (Tablica 5.1). Tablica se sastoji od 4 redaka i 6 stupaca. U prvom retku zapisane su veličine i podaci koje se unose za svaku od metoda. U drugom retku zapisane su veličine i podaci GA metode, u trećem retku veličine i podaci PSO metode te u zadnjem retku veličine i podaci ABC metode.

Tablica 5.1 Rezultati svih metoda

Metoda	Iznos funkcije cilja [mm]	*m*	*U* [kV]	Postignuto nakon koliko iteracija	Postignuti kriterij zaustavljanja
GA metoda	**0,041579**	**2,81**	**50**	**86**	**FunTol**
PSO metoda	**0,041579**	**2,81**	**50**	**4**	**MaxIt**
ABC metoda	**0,041579**	**2,81**	**50**	**1**	**MaxIt**

Kako se može vidjeti s prethodne tablice, rezultati svih triju metoda se ni malo ne razlikuju u pogledu završnog rješenja. Sa svim metodama postignuta je ista vrijednost funkcije cilja, što automatski znači da su postignute i iste vrijednosti nezavisnih varijabli *m* i *U*. Jedino po čemu se razlikuju rezultati metoda je u tome što je GA metodi bilo potrebnije znatno više iteracija od

preostalih dviju metoda, kako bi postigla najbolju vrijednost funkcije cilja, te u tome što se za GA metodu koristilo više različitih kriterija zaustavljanja, dok za preostale dvije metode samo jedan. Ta razlika, u broju iteracija, mogla bi se smanjiti dodatnim promjenama parametara GA metode. Prvenstveno, namještanjem da je jedan od položaja početne populacije GA metode na gornjim vrijednostima nezavisnih varijabli, čime bi postigli pronalazak najbolje vrijednosti funkcije cilja već u prvoj iteraciji. Što se tiče razlike u kriterijima zaustavljanja, PSO i ABC algoritam je zaustavljen postizanjem maksimalnog broja iteracija (100), dok GA algoritam je prestao s pretraživanjem zbog toga što je prosječna promjena iznosa funkcije cilja bila manja od zadane vrijednosti (10^{-10}).

ZAKLJUČAK

Opće je poznato da metaheurističke metode optimizacije ne garantiraju pronalazak optimalnog rješenja, kao što to garantiraju egzaktne metode, već pronalaze rješenje koje se nalazi blizu optimalnog rješenja. No, ono što karakterizira metaheurističke metode je pronalazak rješenja u razumnom vremenskom roku, dok egzaktne metode ne vode računa o utrošku vremena. Iako se za metaheurističke metode tvrdi da ne postoji garancija bliskosti optimuma za rješenja koja pronalaze, može se sa sigurnošću reći da to ovdje nije bio taj slučaj. Nakon implementacije i provedbe GA, PSO i ABC metode najmanje postignuto odstupanje CT vrijednosti promjera vanjskog cilindra od referentne vrijednosti je Δd_v = 0,0416 mm, a to je postignuto za iznos faktora geometrijskog povećanja m = 2,81 te za napon izvora rendgenskih zraka U = 50 kV. Usporedbom dobivenih rezultata s jednadžbom funkcije cilja i danim ograničenjima u pogledu gornje i donje granice nezavisnih varijabli m i U, jednostavno se utvrđuje da su sve tri metode postigle globalni optimum. To je utvrđeno tako što je najmanje odstupanje CT vrijednosti promjera vanjskog cilindra od referentne vrijednosti postignuto za vrijednosti nezavisnih varijabli koje se nalaze na svojim gornjim granicama intervala (domena), tj. vrijednosti nezavisnih varijabli se ne razlikuju od svojih gornjih ograničenja baš u niti jednoj decimali. Stoga, sa sigurnošću možemo tvrditi da se radi o globalnom minimumu jer je rezultat postignut na gornjim granicama varijabli. Osim toga, metode koje su implementirane za rješavanje ovog problema optimizacije imaju sposobnost pronalaska globalnog optimuma. Stoga, zaključuje se da su metaheurističke metode optimizacije itekako pogodne za rješavanje ovog problema optimizacije.

LITERATURA

[1] Molnar, Goran. *Heuristički i metaheuristički postupci optimizacije*. Zagreb. (pristupljeno 5. listopada 2016.) https://www.fer.unizg.hr/_download/repository/Goran_Molnar_KDI.pdf

[2] Raju. 2014. *Optimization methods for engineers*. PHI Learning. New Delhi.

[3] Talbi, El-Ghazali. 2009. *Metaheuristics - From design to implementation*. John Wiley & Sons. Hoboken.

[4] Čajko, Damir. 2013. *Optimiranje tehnoloških procesa metodama nelinearnog programiranja*. Diplomski rad. Fakultet strojarstva i brodogradnje Sveučilišta u Zagrebu. Zagreb. 1 str.

[5] Marti, Rafael; Reinelt, Gerhard. 2011. *The linear ordering problem*. Springer-Verlag GmbH. Berlin.

[6] Gudlin, Mihael. *Heurističke metode*. Operacijska istraživanja II. Fakultet strojarstva i brodogradnje. Zagreb. [predavanje]

[7] *Flowchart maker and online diagram software*. https://www.draw.io/ [On-line softver]

[8] Yang, Xin-She i dr. 2013. *Metaheuristic applications in structures and infrastructures*. Newnes. Oxford.

[9] Stefanoiu, Dan i dr. 2014. *Optimization in engineering sciences*. John Wiley & Sons. New Jersey. London.

[10] Wikipedia. *Six sigma*. (pristupljeno 10. listopada 2016.) https://en.wikipedia.org/wiki/Six_Sigma

[11] Mitrović-Vargra, Zorana. 2008. *Metaheuristike*. Magistarski rad. Matematički fakultet Sveučilišta u Beogradu. Beograd. 28 str.

[12] Glover, Fred; Laguna, Manuel. *Tabu search*. Boston. (pristupljeno 10. listopada 2016.) https://pdfs.semanticscholar.org/0b5d/1e753e65415b6bb441c0db7ed47354187b49.pdf

[13] Stanković, Miloš. 2013. *Rješavanje nekih problema kombinatorne optimizacije algoritmom tabu pretraživanja*. Magistarski rad. Matematički fakultet Sveučilišta u Beogradu. Beograd. 10 str.

[14] Stidsen, Thomas. *GRASP - A speedy introduction*. DTU-Management. Technical University of Denmark. [predavanje]

[15] Crnjac-Milić, Dominika; Masle, Dino. 2013. *Mogućnost primjene monte carlo metode na primjeru agroekonomskog problema prilikom donošenja odluka u uvjetima rizika*. Sveučilište Josipa Jurja Strossmayera u Osijeku. Osijek.

[16] Golub, Marin. *Genetski algoritam*. Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu. Zagreb. (pristupljeno 21. listopada 2016.) http://www.zemris.fer.hr/~golub/ga/ga.html

[17] Bradvica, Vinko. *Algoritmi koji oponašaju procese u prirodi*. Seminarski rad. Fakultet elektrotehnike i računarstva Sveučilišta u Zagrebu. Zagreb.

[18] *Roulette wheel selection*. Newcastle Univerity. Newcastle. (pristupljeno 25. listopada 2016.) http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php

[19] *IX. Selection.* (pristupljeno 25. listopada 2016.) http://www.obitko.com/tutorials/genetic-algorithms/selection.php

[20] Rozenberg, Grzegorz. 2005. *Theoretical computer science*. Elsevier. Amsterdam.

[21] *An ant's rant.* (pristupljeno 3. studenog 2016.) http://docudharma.com/2010/04/an-ants-rant

[22] Parsopoulos, Konstantinos; Vrahatis, Michael. 2010. *Particle swarm optimization and intelligence: Advances and applications*. IGI Global. Janjina (Grčka).

[23] Hu Xiaohui. 2006. *Particle swarm optimization*. (pristupljeno 10. studenog 2016.) http://www.swarmintelligence.org/

[24] Tedorović, Dušan; Šelmić Milica; Davidović Tatjana. 2013. *Bee colony optimization part II: The application survey*. Yugoslav Journal of Operations Research. Beograd. http://elib.mi.sanu.ac.rs/files/journals/yjor/52/yujorn53p185-219.PDF

[25] Karaboga, Dervis; Akay, Bahriye. 2009. *A comparative study of artificial bee colony algorithm*. Kayseri (Turska). (pristupljeno 11. studenog 2016.)

[26] *Computed Tomography (CT) - Body*. (pristupljeno 12. studenog 2016.) http://www.radiologyinfo.org/en/info.cfm?pg=bodyct

[27] *Industrial computed tomography*. 2015. Wikipedia. (pristupljeno 12. studenog 2016.) https://en.wikipedia.org/wiki/Industrial_computed_tomography

[28] Horvatić Novak, Amalija. 2016. *Djelomični plan pokusa*. Seminarski rad. Fakultet strojarstva i brodogradnje Sveučilišta u Zagrebu. Zagreb.

PRILOZI

I. CD-R disk